

UNIT – IV

Syllabus part-1: HADOOP ECO-SYSTEM: Big Data Analytics - Demos, Hadoop and the Amazon Cloud, Query languages for Hadoop, Spreadsheet-like analytics, Stream Computing

Apache Hadoop on Amazon EMR

Apache™ Hadoop® is an open source software project that can be used to efficiently process large datasets. Instead of using one large computer to process and store the data, Hadoop allows clustering commodity hardware together to analyze massive data sets in parallel.

There are many applications and execution engines in the Hadoop ecosystem, providing a variety of tools to match the needs of your analytics workloads. Amazon EMR makes it easy to create and manage fully configured, elastic clusters of Amazon EC2 instances running Hadoop and other applications in the Hadoop ecosystem.

Applications and frameworks in the Hadoop ecosystem

Hadoop commonly refers to the actual Apache Hadoop project, which includes MapReduce (execution framework), YARN (resource manager), and HDFS (distributed storage). You can also install Apache Tez, a next-generation framework which can be used instead of Hadoop MapReduce as an execution engine. Amazon EMR also includes EMRFS, a connector allowing Hadoop to use Amazon S3 as a storage layer.

However, there are also other applications and frameworks in the Hadoop ecosystem, including tools that enable low-latency queries, GUIs for interactive querying, a variety of interfaces like SQL, and distributed NoSQL databases. The Hadoop ecosystem includes many open source tools designed to build additional functionality on Hadoop core components, and you can use Amazon EMR to easily install and configure tools such as Hive, Pig, Hue, Ganglia, Oozie, and HBase on your cluster. You can also run other frameworks, like Apache Spark for in-memory processing, or Presto for interactive SQL, in addition to Hadoop on Amazon EMR.

Hadoop: the basic components

Amazon EMR programmatically installs and configures applications in the Hadoop project, including Hadoop MapReduce, YARN, HDFS, and Apache Tez across the nodes in your cluster.

Processing with Hadoop MapReduce, Tez, and YARN

Hadoop MapReduce and Tez, execution engines in the Hadoop ecosystem, process workloads using frameworks that break down jobs into smaller pieces of work that can be distributed across nodes in your Amazon EMR cluster. They are built with the expectation that any given machine in your cluster could fail at any time and are designed for fault tolerance. If a server running a task fails, Hadoop reruns that task on another machine until completion.

You can write MapReduce and Tez programs in Java, use Hadoop Streaming to execute custom scripts in a parallel fashion, utilize Hive and Pig for higher level abstractions over MapReduce and Tez, or other tools to interact with Hadoop.

Starting with Hadoop 2, resource management is managed by Yet Another Resource Negotiator (YARN). YARN keeps track of all the resources across your cluster, and it

ensures that these resources are dynamically allocated to accomplish the tasks in your processing job. YARN is able to manage Hadoop MapReduce and Tez workloads as well as other distributed frameworks such as Apache Spark.

Storage using Amazon S3 and EMRFS

By using the EMR File System (EMRFS) on your Amazon EMR cluster, you can leverage Amazon S3 as your data layer for Hadoop. Amazon S3 is highly scalable, low cost, and designed for durability, making it a great data store for big data processing. By storing your data in Amazon S3, you can decouple your compute layer from your storage layer, allowing you to size your Amazon EMR cluster for the amount of CPU and memory required for your workloads instead of having extra nodes in your cluster to maximize on-cluster storage. Additionally, you can terminate your Amazon EMR cluster when it is idle to save costs, while your data remains in Amazon S3.

EMRFS is optimized for Hadoop to directly read and write in parallel to Amazon S3 performantly, and can process objects encrypted with Amazon S3 server-side and client-side encryption. EMRFS allows you to use Amazon S3 as your data lake, and Hadoop in Amazon EMR can be used as an elastic query layer.

On-cluster storage with HDFS

Hadoop also includes a distributed storage system, the Hadoop Distributed File System (HDFS), which stores data across local disks of your cluster in large blocks. HDFS has a configurable replication factor (with a default of 3x), giving increased availability and durability. HDFS monitors replication and balances your data across your nodes as nodes fail and new nodes are added.

HDFS is automatically installed with Hadoop on your Amazon EMR cluster, and you can use HDFS along with Amazon S3 to store your input and output data. You can easily encrypt HDFS using an Amazon EMR security configuration. Also, Amazon EMR configures Hadoop to use HDFS and local disk for intermediate data created during your Hadoop MapReduce jobs, even if your input data is located in Amazon S3.

Advantages of Hadoop on Amazon EMR

Increased speed and agility

You can initialize a new Hadoop cluster dynamically and quickly, or add servers to your existing Amazon EMR cluster, significantly reducing the time it takes to make resources available to your users and data scientists. Using Hadoop on the AWS platform can dramatically increase your organizational agility by lowering the cost and time it takes to allocate resources for experimentation and development.

Reduced administrative complexity

Hadoop configuration, networking, server installation, security configuration, and ongoing administrative maintenance can be a complicated and challenging activity. As a managed service, Amazon EMR addresses your Hadoop infrastructure requirements so you can focus on your core business.

Integration with other AWS services

You can easily integrate your Hadoop environment with other services such as Amazon S3, Amazon Kinesis, Amazon Redshift, and Amazon DynamoDB to enable data movement, workflows, and analytics across the many diverse services on the AWS platform. Additionally, you can use the AWS Glue Data Catalog as a managed metadata repository for Apache Hive and Apache Spark.

Pay for clusters only when you need them

Many Hadoop jobs are spiky in nature. For instance, an ETL job can run hourly, daily, or monthly, while modeling jobs for financial firms or genetic sequencing may occur only a few times a year. Using Hadoop on Amazon EMR allows you to spin up these workload clusters easily, save the results, and shut down your Hadoop resources when they're no longer needed, to avoid unnecessary infrastructure costs. EMR 6.x supports Hadoop 3, which allows the YARN NodeManager to launch containers either directly on the EMR cluster host or inside a Docker container. Please see our documentation to learn more.

Improved availability and disaster recovery

By using Hadoop on Amazon EMR, you have the flexibility to launch your clusters in any number of Availability Zones in any AWS region. A potential problem or threat in one region or zone can be easily circumvented by launching a cluster in another zone in minutes.

Flexible capacity

Capacity planning prior to deploying a Hadoop environment can often result in expensive idle resources or resource limitations. With Amazon EMR, you can create clusters with the required capacity within minutes and use EMR Managed Scaling to dynamically scale out and scale in nodes.

Query languages for Hadoop:

If you want a high-level query language for drilling into your huge Hadoop dataset, then you've got some choice:

- Pig, from Yahoo! and now incubating at Apache, has an imperative language called Pig Latin for performing operations on large data files.
- Jaql, from IBM and soon to be open sourced, is a declarative query language for JSON data.
- Hive, from Facebook and soon to become a Hadoop contrib module, is a data warehouse system with a declarative query language that is a hybrid of SQL and Hadoop streaming.

All three projects have different strengths, but there is plenty of scope for collaboration and cross-pollination, particularly in the query language.

Syllabus part-2: Pig: Introduction to PIG, Execution Modes of Pig, Comparison of Pig with Databases, Grunt, Pig Latin, User Defined Functions, Data Processing operators.

1. What is Apache PIG? Explain various benefits of Apache PIG.

Ans: Pig is a high-level data flow platform for executing Map Reduce programs of Hadoop. It was developed by Yahoo. The language for Pig is pig Latin.

Features of Apache Pig

Let's see the various uses of Pig technology.

- 1) **Ease of programming:** Writing complex java programs for map reduce is quite tough for non-programmers. Pig makes this process easy. In the Pig, the queries are converted to MapReduce internally.
- 2) **Optimization opportunities:** It is how tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.
- 3) **Extensibility:** A user-defined function is written in which the user can write their logic to execute over the data set.
- 4) **Flexible:** It can easily handle structured as well as unstructured data.
- 5) **In-built operators:** It contains various type of operators such as sort, filter and joins.
- 6) **Less code** - The Pig consumes less line of code to perform any operation.
- 7) **Reusability** - The Pig code is flexible enough to reuse again.
- 8) **Nested data types** - The Pig provides a useful concept of nested data types like tuple, bag, and map.

2. Differentiate between MapReduce and PIG.

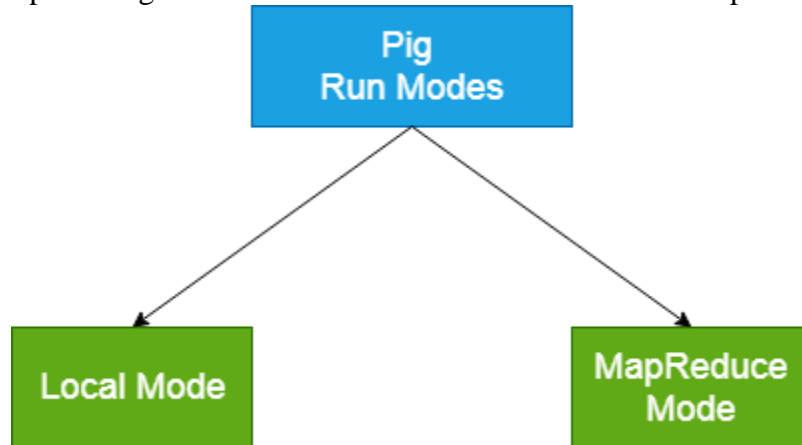
Ans:Differences between Apache MapReduce and PIG

Apache MapReduce		Apache PIG
It is a low-level data processing tool.		It is a high-level data flow tool.
Here, it is required to develop complex programs using Java or Python.		It is not required to develop complex programs.
It is difficult to perform data operations in MapReduce.		It provides built-in operators to perform data operations like union, sorting and ordering.
It doesn't allow nested data types.		It provides nested data types like tuple, bag, and map.
Basis for comparison	PIG	MapReduce
Operations	<ul style="list-style-type: none">• Dataflow language.• High-Level Language.• Performing join operations in a pig are simple	<ul style="list-style-type: none">• Data processing language.• Low-level language• Quite difficult to perform the join operations.
Lines of code and verbosity	Multi-query approach, thereby reducing the length of the codes.	require almost 10 times more the number of lines to perform the same task.
Compilation	No need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job.	MapReduce jobs have a long compilation process.
Code portability	Works with any of the versions in Hadoop	No guarantee that supports with every version in Hadoop

3. Explain the various PIG running modes and various ways to execute the PIG programs.

Ans: Apache Pig Run Modes

Apache Pig executes in two modes: Local Mode and MapReduce Mode.



Local Mode

- It executes in a single JVM and is used for development experimenting and prototyping.
- Here, files are installed and run using localhost.
- The local mode works on a local file system. The input and output data stored in the local file system.

The command for local mode grunt shell:

```
$ pig-x local
```

MapReduce Mode

- The MapReduce mode is also known as Hadoop Mode.
- It is the default mode.
- In this Pig renders Pig Latin into MapReduce jobs and executes them on the cluster.
- It can be executed against semi-distributed or fully distributed Hadoop installation.
- Here, the input and output data are present on HDFS.

The command for Map reduce mode:

```
$ pig
```

Or,

```
$ pig -x mapreduce
```

Ways to execute Pig Program

These are the following ways of executing a Pig program on local and MapReduce mode: -

- **Interactive Mode** - In this mode, the Pig is executed in the Grunt shell. To invoke Grunt shell, run the pig command. Once the Grunt mode executes, we can provide Pig Latin statements and command interactively at the command line.
- **Batch Mode** - In this mode, we can run a script file having a .pig extension. These files contain Pig Latin commands.
- **Embedded Mode** - In this mode, we can define our own functions. These functions can be called as UDF (User Defined Functions). Here, we use programming languages like Java and Python.

4. Explain the steps involved to create and execute the PIG user defined functions.

Ans: Pig UDF (User Defined Functions)

To specify custom processing, Pig provides support for user-defined functions (UDFs). Thus, Pig allows us to create our own functions. Currently, Pig UDFs can be implemented using the following programming languages: -

- Java
- Python
- Jython
- JavaScript

- Ruby
- Groovy

Among all the languages, Pig provides the most extensive support for Java functions. However, limited support is provided to languages like Python, Jython, JavaScript, Ruby, and Groovy.

Example of Pig UDF

In Pig,

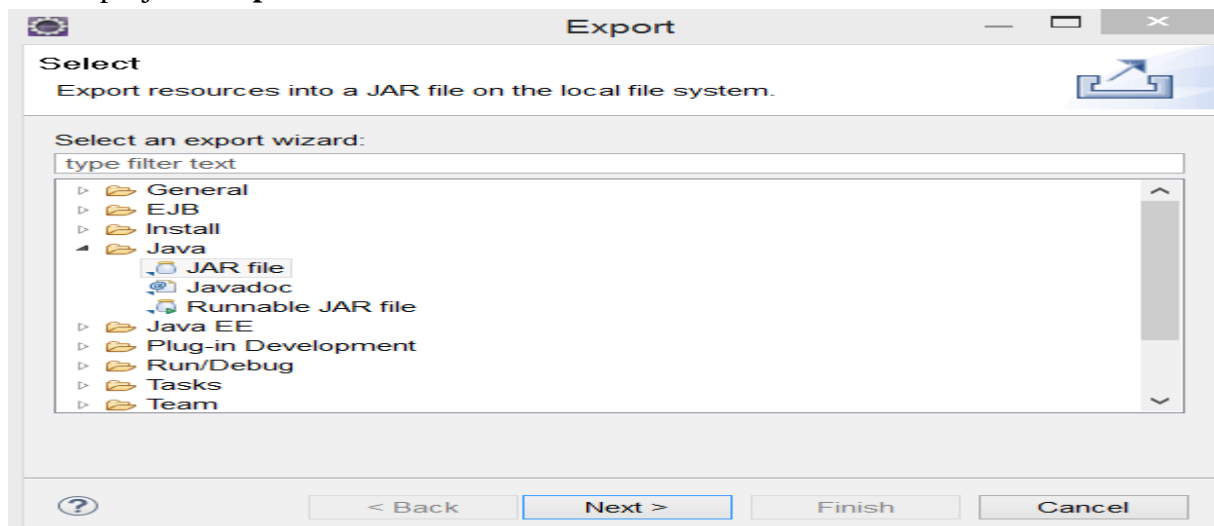
- All UDFs must extend "org.apache.pig.EvalFunc"
- All functions must override the "exec" method.

Let's see an example of a simple EVAL Function to convert the provided string to uppercase.

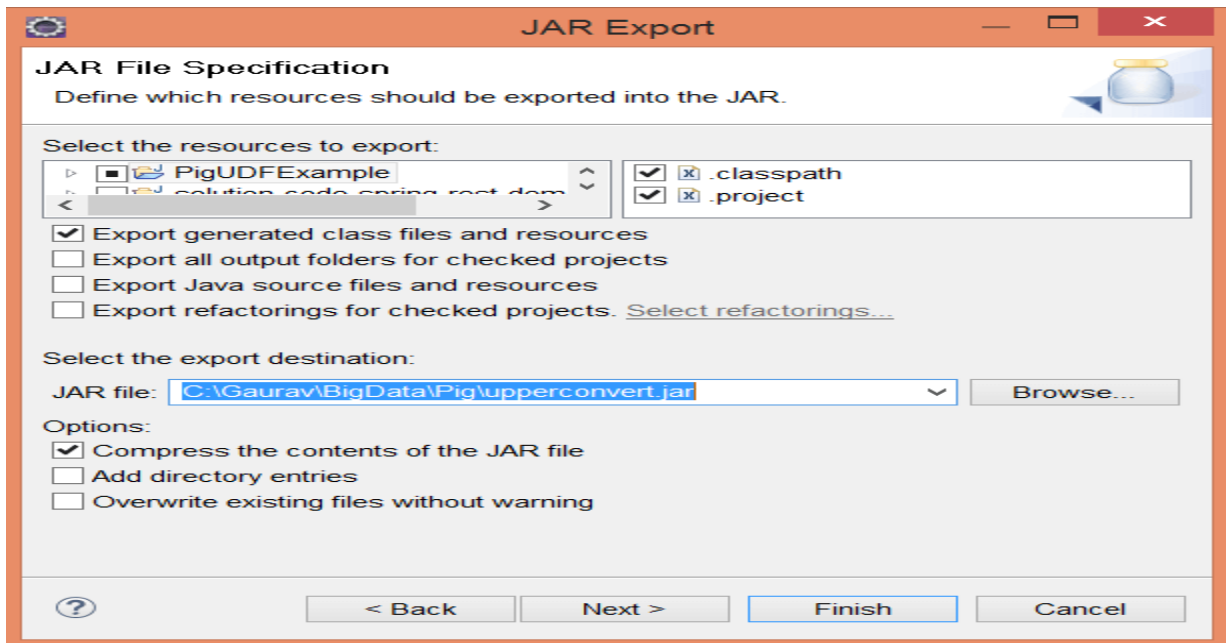
UPPER.java

```
package com.hadoop;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
public class TestUpper extends EvalFunc<String> {
    public String exec(Tuple input) throws IOException {
        if (input == null || input.size() == 0)
            return null;
        try{
            String str = (String)input.get(0);
            return str.toUpperCase();
        }catch(Exception e){
            throw new IOException("Caught exception processing input row ", e);
        }
    }
}
```

- Create the jar file and export it into the specific directory. For that ,right click on project - **Export - Java - JAR file - Next.**

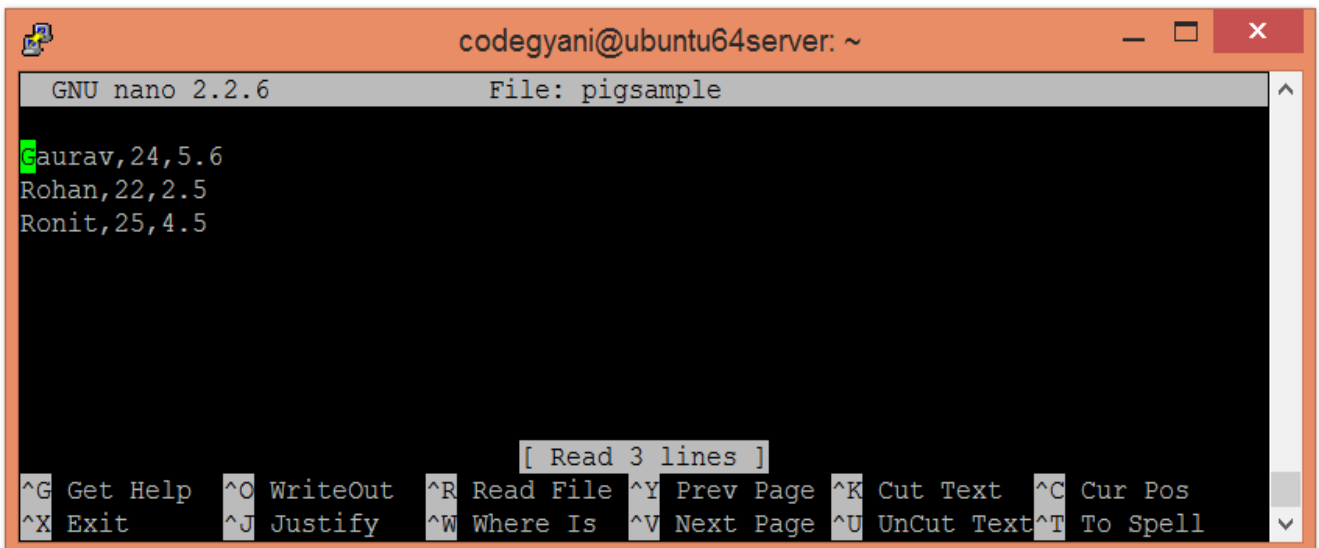


- Now, provide a specific name to the jar file and save it in a local system directory.



- Create a text file in your local machine and insert the list of tuples.

1. \$ nano pignsample

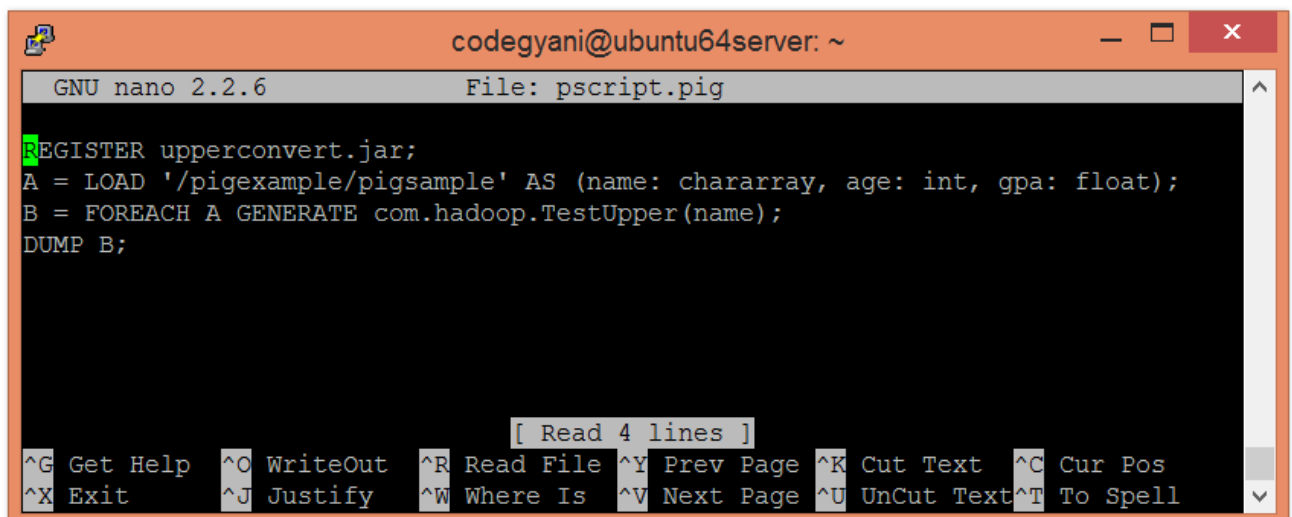


- Upload the text files on HDFS in the specific directory.

1. \$ hdfs dfs -put pigexample /pigexample

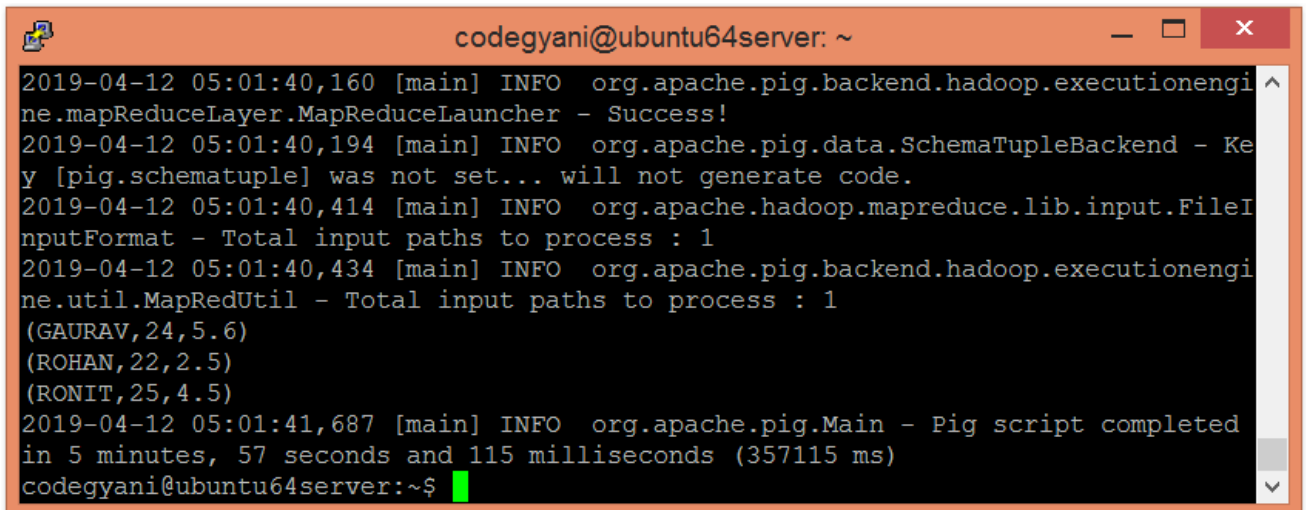
- Create a pig file in your local machine and write the script.

1. \$ nano pscript.pig



- Now, run the script in the terminal to get the output.

1. \$pig pscript.pig



```
codegyani@ubuntu64server: ~
2019-04-12 05:01:40,160 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-04-12 05:01:40,194 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-04-12 05:01:40,414 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-04-12 05:01:40,434 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(GAURAV,24,5.6)
(ROHAN,22,2.5)
(RONIT,25,4.5)
2019-04-12 05:01:41,687 [main] INFO org.apache.pig.Main - Pig script completed in 5 minutes, 57 seconds and 115 milliseconds (357115 ms)
codegyani@ubuntu64server:~$
```

Here, we got the desired output.

5. Explain the following data processing operators with examples.

i.LOAD

ii.CROSS

iii.DISTINCT

iv.FILTER

Ans:Apache Pig LOAD Operator: The Apache Pig LOAD operator is used to load the data from the file system.

Syntax

LOAD 'info' [USING FUNCTION] [AS SCHEMA];

Here,

- o **LOAD** is a relational operator.
- o **'info'** is a file that is required to load. It contains any type of data.
- o **USING** is a keyword.
- o **FUNCTION** is a load function.
- o **AS** is a keyword.
- o **SCHEMA** is a schema of passing file, enclosed in parentheses.

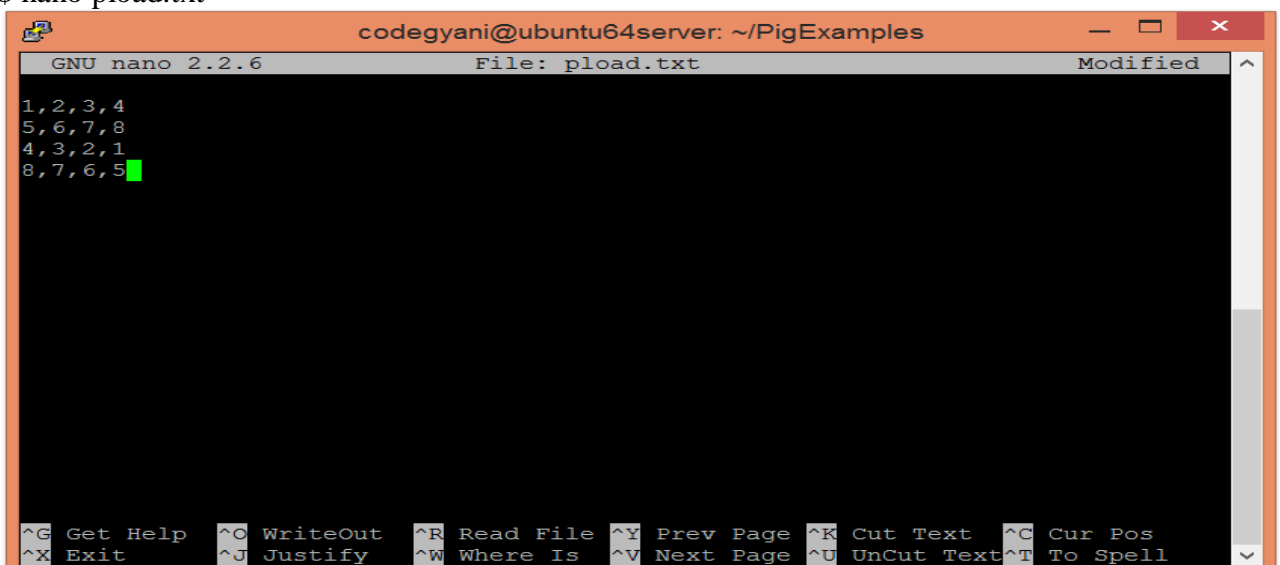
Example of LOAD Operator

In this example, we load the text file data from the file system.

Steps to execute LOAD Operator

- o Create a text file in your local machine and provide some values to it.

\$ nano pload.txt



```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: pload.txt Modified
1,2,3,4
5,6,7,8
4,3,2,1
8,7,6,5
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- o Check the values written in the text files.

\$ cat pload.txt


```
codegyani@ubuntu64server: ~/PigExamples
codegyani@ubuntu64server:~/PigExamples$ nano pload.txt
codegyani@ubuntu64server:~/PigExamples$ cat pload.txt
1,2,3,4
5,6,7,8
4,3,2,1
8,7,6,5
codegyani@ubuntu64server:~/PigExamples$
```

- Upload the text files on HDFS in the specific directory.
- ```
$ hdfs dfs -put pload.txt /pigexample
```
- Open the pig MapReduce run mode.
- ```
$ pig
```
- Load the file that contains the data.
- ```
grunt> A = LOAD '/pigexample/pload.txt' USING PigStorage(',') AS (a1:int,a2:int,a3:int,a4:
int);
```
- Now, execute and verify the data.
- ```
grunt> DUMP A;
```

Apache Pig CROSS Operator: The Apache Pig CROSS operator facilitates to compute the cross product of two or more relations. Using CROSS operator is an expensive operation and should be used sparingly.

Example of CROSS Operator

In this example, we compute the data of two relations.

Steps to execute CROSS Operator

- Create a text file in your local machine and write some values into it.
- ```
$ nano pcross1.txt
```

```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: pcross1.txt
2,5
3,6
[Read 2 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Create another text file in your local machine and also write some values into it.
- ```
$ nano pcross2.txt
```

```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: pcross2.txt
3,6,8
2,6,9
[ Read 2 lines ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the values written in both text files.

```
$ cat pcross1.txt
```

```
$ cat pcross2.txt
```

```
codegyani@ubuntu64server: ~/PigExamples
codegyani@ubuntu64server:~/PigExamples$ cat pcross1.txt
2,5
3,6
codegyani@ubuntu64server:~/PigExamples$ cat pcross2.txt
3,6,8
2,6,9
codegyani@ubuntu64server:~/PigExamples$
```

- Upload both text files on HDFS in the specific directory.

```
$ hdfs dfs -put pcross1.txt /pigexample
```

```
$ hdfs dfs -put pcross2.txt /pigexample
```

- Open the pig MapReduce run mode.

```
$ pig
```

- Load the file that contains the data.

```
grunt> A = LOAD '/pigexample/pcross1.txt' USING PigStorage(',') AS (a1:int,a2:int);
```

- Now, execute and verify the data.

```
grunt> DUMP A;
```

- Load the another file that contains the data.

```
grunt> B = LOAD '/pigexample/pcross2.txt' USING PigStorage(',') AS (b1:int,b2:int,b3:int);
```

- Now, execute and verify the data.

```
grunt> DUMP B;
```

- Let's perform the Cartesian product between both files.

```
grunt> Result = CROSS A,B;
```

- Now, execute and verify the data.

grunt> DUMP Result;

```
codegyani@ubuntu64server: ~/PigExamples
RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-25 01:33:25,171 [main] INFO  org.apache.hadoop.ipc.Client - Retrying con
nect to server: 0.0.0.0/0.0.0.0:10020. Already tried 9 time(s); retry policy is
RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-25 01:33:25,276 [main] WARN  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-25 01:33:25,277 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 01:33:25,326 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-25 01:33:25,446 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-25 01:33:25,450 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(3,6,2,6,9)
(3,6,3,6,8)
(2,5,2,6,9)
(2,5,3,6,8)
grunt>
```

```
codegyani@ubuntu64server: ~/PigExamples
aggregation.
2019-02-25 07:18:49,966 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 07:18:50,007 [main] WARN  org.apache.pig.data.SchemaTupleBackend - Sc
hemaTupleBackend has already been initialized
2019-02-25 07:18:50,176 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-25 07:18:50,181 [main] INFO  org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(1,2,3,4)
(5,6,7,8)
(4,3,2,1)
(8,7,6,5)
grunt>
```

- o Let's check the corresponding schema.

grunt> DESCRIBE A;

```
codegyani@ubuntu64server: ~/PigExamples
grunt> DESCRIBE A;
A: {a1: int,a2: int,a3: int,a4: int}
grunt>
```

Apache Pig DISTINCT Operator: The Apache Pig DISTINCT operator is used to remove duplicate tuples in a relation. Initially, Pig sorts the given data and then eliminates duplicates.

Example of DISTINCT Operator

In this example, we eliminate the duplicate tuples.

Steps to execute DISTINCT Operator

- o Create a text file in your local machine and provide some values to it.

\$ nano pdistinct.txt

```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: pdistinct.txt Modified
1,3,5
2,1,4
1,3,5
1,4,2
2,1,4
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the values written in the text files.

\$ cat pdistinct.txt

```
codegyani@ubuntu64server: ~/PigExamples
codegyani@ubuntu64server:~/PigExamples$ cat pdistinct.txt
1,3,5
2,1,4
1,3,5
1,4,2
2,1,4
codegyani@ubuntu64server:~/PigExamples$
```

- Upload the text files on HDFS in the specific directory.

\$ hdfs dfs -put pdistinct.txt /pigexample

- Open the pig MapReduce run mode.

\$ pig

- Load the file that contains the data.

grunt> A = LOAD '/pigexample/pdistinct.txt' USING PigStorage(',') as (a1:int,a2:int,a3:int);

- Now, execute and verify the data.

grunt> DUMP A;

```
codegyani@ubuntu64server: ~/PigExamples
aggregation.
2019-02-25 04:44:22,669 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 04:44:22,698 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-25 04:44:22,853 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 04:44:22,857 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(1,3,5)
(2,1,4)
(1,3,5)
(1,4,2)
(2,1,4)
grunt>
```

- Let's execute DISTINCT operator to eliminate duplicate tuples.
- ```
grunt> Result = DISTINCT A;
```
- Now, execute and verify the data.
- ```
grunt> DUMP Result;
```

```
codegyani@ubuntu64server: ~/PigExamples
2019-02-25 04:53:14,595 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-25 04:53:14,598 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 04:53:14,601 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-25 04:53:14,694 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 04:53:14,696 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(1,3,5)
(1,4,2)
(2,1,4)
grunt>
```

Apache Pig FILTER Operator: The Apache Pig FILTER operator is used to remove duplicate tuples in a relation. Initially, Pig sorts the given data and then eliminates duplicates.

Example of FILTER Operator

In this example, we eliminate duplicate tuples.

Steps to execute FILTER Operator

- Create a text file in your local machine and provide some values to it.

```
$ nano pfilter.txt
```

```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: pfilter.txt Modified
1,2
2,8
4,5
9,3
7,8
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the values written in the text files.

\$ cat pfilter.txt

```
codegyani@ubuntu64server: ~/PigExamples
codegyani@ubuntu64server:~/PigExamples$ nano pfilter.txt
codegyani@ubuntu64server:~/PigExamples$ cat pfilter.txt
1,2
2,8
4,5
9,3
7,8
codegyani@ubuntu64server:~/PigExamples$
```

- Upload the text files on HDFS in the specific directory.

\$ hdfs dfs -put pfilter.txt /pigexample

- Open the pig MapReduce run mode.

\$ pig

- Load the file that contains the data.

grunt> A = LOAD '/pigexample/pfilter.txt' USING PigStorage(',') AS (a1:int,a2:int);

- Now, execute and verify the data

grunt> DUMP A;

```
codegyani@ubuntu64server: ~/PigExamples
2019-02-25 06:13:20,029 [main] INFO org.apache.pig.backend.hadoop.executionengine.map
uceLayer.MapReduceLauncher - Success!
2019-02-25 06:13:20,064 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig
hematuple] was not set... will not generate code.
2019-02-25 06:13:20,258 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFo
t - Total input paths to process : 1
2019-02-25 06:13:20,263 [main] INFO org.apache.pig.backend.hadoop.executionengine.uti
apRedUtil - Total input paths to process : 1
(1,2)
(2,8)
(4,5)
(9,3)
(7,8)
grunt>
```

- Let's execute FILTER operator to eliminate duplicate tuples.

grunt> Result = FILTER A BY a2==8;

- Now, execute and verify the data.

grunt> DUMP Result;

```
codegyani@ubuntu64server: ~/PigExamples
2019-02-25 06:24:50,721 [main] INFO org.apache.hadoop.ipc.Client - Retrying connect to server: 0.0.0.0/0.0.0.0:10020. Already tried 9 time(s); retry policy is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-25 06:24:50,854 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-25 06:24:50,860 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 06:24:50,870 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-25 06:24:50,996 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 06:24:50,998 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(2,8)
(7,8)
grunt>
```

6. Explain the following pre-defined PIG functions with examples.

i.AVG

ii.CONCAT

iii.COUNT

iv. IN

Ans:

Apache Pig AVG Function:The Apache Pig AVG function is used to find the average of given numeric values in a single-column bag. It requires a preceding GROUP ALL statement for global averages and a GROUP BY statement for group averages. However, it ignores the NULL values.

Example of AVG Function

In this example, we will compute the average of given numeric values.

Steps to execute AVG Function

- Create a text file in your local machine and insert the list of tuples.

\$ nano evalavg.txt

```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: evalavg.txt Modified
Rohan,a,2.4F
Rohan,b,2.8F
Rohan,c,3.6F
Sam,d,2.2F
Sam,e,3.4F
Sam,f,3.8F
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the tuples inserted in the text files.

\$ cat evalavg.txt

```
codegyani@ubuntu64server: ~
codegyani@ubuntu64server:~$ cat evalavg.txt
Rohan,a,2.4F
Rohan,b,2.8F
Rohan,c,3.6F
Sam,d,2.2F
Sam,e,3.4F
Sam,f,3.8F
codegyani@ubuntu64server:~$
```

- Upload the text files on HDFS in the specific directory.
- ```
$ hdfs dfs -put evalavg.txt /pigexample
```
- Open the pig MapReduce run mode.
- ```
$ pig
```
- Load the file that contains the data.
- ```
grunt> A = LOAD '/pigexample/evalavg.txt' USING PigStorage(',') AS (a1:chararray,a2:chararray,a3:float);
```
- Now, execute and verify the data.
- ```
grunt> DUMP A;
```

```
codegyani@ubuntu64server: ~
RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-26 08:49:52,631 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-26 08:49:52,637 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 08:49:52,729 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 08:49:52,912 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 08:49:52,917 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Rohan,a,2.4)
(Rohan,b,2.8)
(Rohan,c,3.6)
(Sam,d,2.2)
(Sam,e,3.4)
(Sam,f,3.8)
grunt>
```

- Let's group the data on the basis of 'a1' field.
- ```
grunt> B = GROUP A BY a1;
grunt> DUMP B;
```



```
codegyani@ubuntu64server: ~
aggregation.
2019-02-26 08:59:30,531 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 08:59:30,537 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 08:59:30,667 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 08:59:30,672 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Sam, {(Sam, f, 3.8), (Sam, e, 3.4), (Sam, d, 2.2)})
(Rohan, {(Rohan, c, 3.6), (Rohan, b, 2.8), (Rohan, a, 2.4)})
grunt>
```

- Let's return the average of given numeric values.

```
grunt> Result = FOREACH B GENERATE A.a1, AVG(A.a3);
```

```
grunt> DUMP Result;
```

```
codegyani@ubuntu64server: ~
aggregation.
2019-02-26 09:10:00,071 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 09:10:00,078 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 09:10:00,166 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 09:10:00,169 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(({Sam}), (Sam), (Sam)), 3.133333365122477)
(({Rohan}), (Rohan), (Rohan)), 2.9333333174387612)
grunt>
```

**Apache Pig CONCAT Function:** The Apache Pig CONCAT function is used to concatenate two or more expressions. The generated result of expression must have identical types. However, if any sub-expression is null, the generated expression is also null.

Example of CONCAT Function

In this example, we concatenate the first two fields of each tuple.

Steps to execute CONCAT Function

- Create a text file in your local machine and insert the list of tuples.

```
$ nano evalconcat.txt
```

```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: evalconcat.txt Modified
Jason,Roy,27
Peter,Holder,25
William,Hope,30
Roman,Naughtan,22
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the tuples inserted in the text files.

\$ cat evalconcat.txt

```
codegyani@ubuntu64server: ~$ nano evalconcat.txt
codegyani@ubuntu64server:~$ cat evalconcat.txt
Jason,Roy,27
Peter,Holder,25
William,Hope,30
Roman,Naughtan,22
codegyani@ubuntu64server:~$
```

- Upload the text files on HDFS in the specific directory.

\$ hdfs dfs -put evalconcat.txt /pigexample

- Open the pig MapReduce run mode.

\$ pig

- Load the file that contains the data.

```
grunt> A = LOAD '/pigexample/evalconcat.txt' USING PigStorage(',') AS (a1:chararray,a2:chararray,a3:int);
```

- Now, execute and verify the data.

```
grunt> DUMP A;
```

```
codegyani@ubuntu64server: ~
aggregation.
2019-02-26 10:05:47,224 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 10:05:47,286 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 10:05:47,513 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 10:05:47,521 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Jason,Roy,27)
(Peter,Holder,25)
(William,Hope,30)
(Roman,Naughtan,22)
grunt>
```

- Let's return the concatenation of two fields of each tuple.

Result = FOREACH A GENERATE CONCAT (a1,'\_',a2);

DUMP Result;

```
codegyani@ubuntu64server: ~
2019-02-27 05:30:45,487 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-27 05:30:45,489 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 05:30:45,498 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-27 05:30:45,577 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-27 05:30:45,578 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Jason_Roy)
(Peter_Holder)
(William_Hope)
(Roman_Naughtan)
grunt>
```

**Apache Pig COUNT Function:**The Apache Pig COUNT function is used to count the number of elements in a bag. It requires a preceding GROUP ALL statement for global counts and a GROUP BY statement for group counts. It ignores the null values.

Example of COUNT Function

In this example, we count the tuples in the bag.

Steps to execute COUNT Function

- Create a text file in your local machine and insert the list of tuples.

\$ nano evalcount.txt

```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: evalcount.txt Modified
5,2,3
1,3,6
5,1,2
3,8,9
1,3,7
7,2,5
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the tuples inserted in the text files.

\$ cat evalcount.txt

```
codegyani@ubuntu64server: ~$ cat evalcount.txt
5,2,3
1,3,6
5,1,2
3,8,9
1,3,7
7,2,5
codegyani@ubuntu64server: ~$
```

- Upload the text files on HDFS in the specific directory.

\$ hdfs dfs -put evalcount.txt /pigexample

- Open the pig MapReduce run mode.

\$ pig

- Load the file that contains the data.

```
grunt> A = LOAD '/pigexample/evalcount.txt' USING PigStorage(',') AS (a1:int,a2:int,a3:int)
;
```

- Now, execute and verify the data.

```
grunt> DUMP A;
```

```
codegyani@ubuntu64server: ~
aggregation.
2019-02-26 23:38:07,506 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 23:38:07,565 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-26 23:38:07,781 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-26 23:38:07,790 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(5,2,3)
(1,3,6)
(5,1,2)
(3,8,9)
(1,3,7)
(7,2,5)
grunt>
```

- Let's group the data on the basis of 'a1' field.

```
grunt> B = GROUP A BY a1;
```

```
grunt> DUMP B;
```

```
codegyani@ubuntu64server: ~
2019-02-26 23:47:07,059 [main] WARN org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-26 23:47:07,059 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 23:47:07,061 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-26 23:47:07,098 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-26 23:47:07,104 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(1,{(1,3,7),(1,3,6)})
(3,{(3,8,9)})
(5,{(5,1,2),(5,2,3)})
(7,{(7,2,5)})
grunt>
```

- Let's return the count of given tuples.

```
grunt> Result = FOREACH B GENERATE COUNT(A);
```

```
grunt> DUMP Result;
```

```
codegyani@ubuntu64server: ~
2019-02-26 23:57:13,333 [main] WARN org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-26 23:57:13,337 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 23:57:13,347 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-26 23:57:13,417 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-26 23:57:13,418 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(2)
(1)
(2)
(1)
grunt>
```

**Apache Pig IN Function:** The Apache Pig IN function is used to reduce the requirement for multiple OR conditions. It facilitates to check if the current expression matches with any value exist in a list.

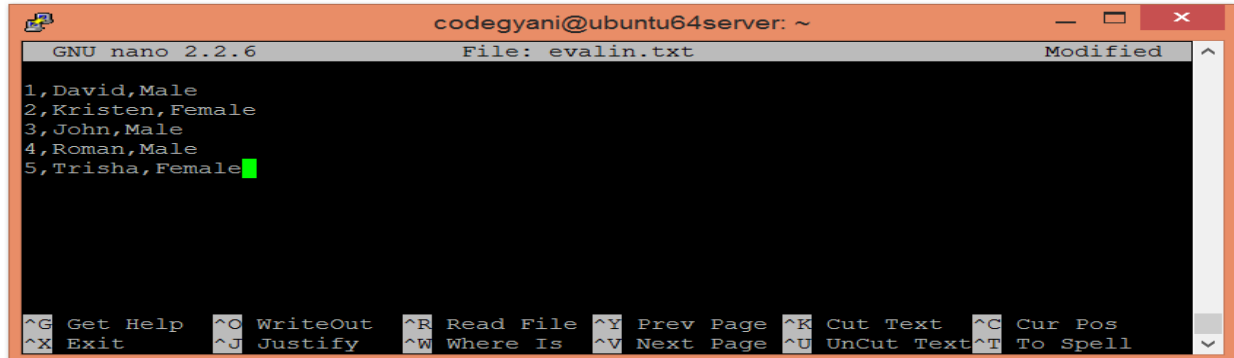
Example of IN Function

In this example, we filter the tuples by applying IN operator on values of the list.

Steps to execute IN Function

- Create a text file in your local machine and insert the list of tuples.

\$ nano evalin.txt



```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: evalin.txt Modified
1,David,Male
2,Kristen,Female
3,John,Male
4,Roman,Male
5,Trisha,Female
Get Help WriteOut Read File Prev Page Cut Text Cur Pos
Exit Justify Where Is Next Page UnCut Text To Spell
```

- Check the tuples inserted in the text files.

\$ cat evalin.txt



```
codegyani@ubuntu64server: ~$ cat evalin.txt
1,David,Male
2,Kristen,Female
3,John,Male
4,Roman,Male
5,Trisha,Female
codegyani@ubuntu64server: ~$
```

- Upload the text files on HDFS in the specific directory.

\$ hdfs dfs -put evalin.txt /pigexample

- Open the pig MapReduce run mode.

\$ pig

- Load the file that contains the data.

```
grunt> A = LOAD '/pigexample/evalin.txt' USING PigStorage(',') AS (a1:int,a2:chararray,a3:chararray);
```

- Now, execute and verify the data.

```
grunt> DUMP A;
```

```
codegyani@ubuntu64server: ~
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-27 00:38:46,555 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 00:38:46,595 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 00:38:46,785 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 00:38:46,789 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(1,David,Male)
(2,Kristen,Female)
(3,John,Male)
(4,Roman,Male)
(5,Trisha,Female)
grunt>
```

- Let's return the filter data.

```
grunt> Result = FILTER A BY a1 IN (2, 4);
grunt> DUMP Result;
```

```
codegyani@ubuntu64server: ~
nect to server: 0.0.0.0/0.0.0.0:10020. Already tried 9 time(s); retry policy is
RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-27 00:59:04,684 [main] WARN org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-27 00:59:04,685 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 00:59:04,690 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 00:59:04,746 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 00:59:04,750 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(2,Kristen,Female)
(4,Roman,Male)
grunt>
```

## 7. Explain the following arithmetic PIG functions with examples.

i.CBRT                      ii.CEIL                      iii.LOG                      iv. ROUND

Ans:

**Apache Pig CBRT Function:** The Apache Pig CBRT function is used to return the cube root of the expression.

**Syntax**

CBRT(expression)

**Example of CBRT Function**

In this example, we find out the cube root of the given value.

**Steps to execute the CBRT Function**

- Create a text file in your local machine and insert the values.  
\$ nano mathcbt.txt



```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: mathcbrt.txt Modified
8
27
64
125
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the values inserted in the text files.

\$ cat mathcbrt.txt

```
codegyani@ubuntu64server: ~$ cat mathcbrt.txt
8
27
64
125
codegyani@ubuntu64server: ~$
```

- Upload the text files on HDFS in the specific directory.  
\$ hdfs dfs -put mathcbrt.txt /pigexample
- Open the pig MapReduce run mode.  
\$ pig
- Load the file that contains the data.  
grunt> A = LOAD '/pigexample/mathcbrt.txt' AS (a1:int) ;
- Now, execute and verify the data.  
grunt> DUMP A;

```
codegyani@ubuntu64server: ~
aggregation.
2019-02-27 12:54:07,485 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 12:54:07,557 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 12:54:07,721 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 12:54:07,724 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(8)
(27)
(64)
(125)
()
grunt>
```

- Let's return the cube root of each value.  
grunt> Result = FOREACH A GENERATE CBRT(a1);  
grunt> DUMP Result;



```
codegyani@ubuntu64server: ~
aggregation.
2019-02-27 13:04:46,113 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 13:04:46,118 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 13:04:46,164 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 13:04:46,168 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(2.0)
(3.0)
(4.0)
(5.0)
()
grunt> █
```

**Apache Pig CEIL Function:** The Apache Pig CEIL function is used to round up the provided value to the nearest integer. It returns the smallest integer greater than or equal to a given value.

### Syntax

CEIL(expression)

### Example of CEIL Function

In this example, we round up each value exist in the file to the nearest integer.

### Steps to execute CEIL Function

- Create a text file in your local machine and insert the values.  
\$ nano mathceil.txt

```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: mathceil.txt Modified
3.2
5.6
4.5
-2.3
-4.5 █

[New File]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the values inserted in the text files.  
\$ cat mathceil.txt

```
codegyani@ubuntu64server: ~
codegyani@ubuntu64server:~$ nano mathceil.txt
codegyani@ubuntu64server:~$ cat mathceil.txt
3.2
5.6
4.5
-2.3
-4.5
codegyani@ubuntu64server:~$ █
```

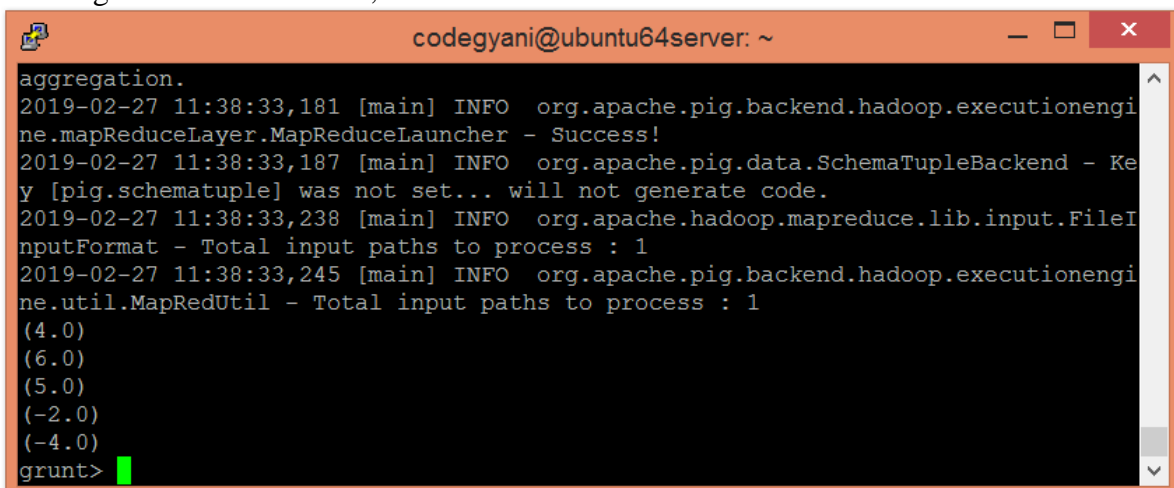
- Upload the text files on HDFS in the specific directory.  
\$ hdfs dfs -put mathceil.txt /pigexample

- Open the pig MapReduce run mode.  
\$ pig
- Load the file that contains the data.  
grunt> A = LOAD '/pigexample/mathceil.txt' AS (a1:float) ;
- Now, execute and verify the data.  
grunt> DUMP A;



```
codegyani@ubuntu64server: ~
aggregation.
2019-02-27 11:28:21,715 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 11:28:21,807 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 11:28:21,981 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 11:28:21,984 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(3.2)
(5.6)
(4.5)
(-2.3)
(-4.5)
grunt>
```

- Let's round up all the values that exist in the text file.  
grunt> Result = FOREACH A GENERATE CEIL(a1);  
grunt> DUMP Result;



```
codegyani@ubuntu64server: ~
aggregation.
2019-02-27 11:38:33,181 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 11:38:33,187 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 11:38:33,238 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 11:38:33,245 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(4.0)
(6.0)
(5.0)
(-2.0)
(-4.0)
grunt>
```

**Apache Pig LOG Function:** The Apache Pig LOG function is used to return the natural logarithm (base e) of an expression.

### Syntax

LOG(expression)

### Example of LOG Function

In this example, we return the log value.

### Steps to execute LOG Function

- Create a text file in your local machine and insert the values.  
\$ nano mathlog.txt

```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: mathlog.txt Modified
10
100
3.2
5.6
4.5
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the values inserted in the text files.  
\$ cat mathlog.txt

```
codegyani@ubuntu64server: ~$ cat mathlog.txt
10
100
3.2
5.6
4.5
codegyani@ubuntu64server: ~$
```

- Upload the text files on HDFS in the specific directory.  
\$ hdfs dfs -put mathlog.txt /pigexample
- Open the pig MapReduce run mode.  
\$ pig
- Load the file that contains the data.  
grunt> A = LOAD '/pigexample/mathlog.txt' AS (a1:float) ;
- Now, execute and verify the data.  
grunt> DUMP A;

```
codegyani@ubuntu64server: ~
2019-04-06 06:46:52,114 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-04-06 06:46:52,120 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-04-06 06:46:52,209 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-04-06 06:46:52,399 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-04-06 06:46:52,412 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(10.0)
(100.0)
(3.2)
(5.6)
(4.5)
grunt>
```

- Let's return the log value.  
grunt> Result = FOREACH A GENERATE LOG(a1);  
grunt> DUMP Result;

```
codegyani@ubuntu64server: ~
aggregation.
2019-04-06 06:55:38,955 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-04-06 06:55:38,960 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-04-06 06:55:39,046 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-04-06 06:55:39,048 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(2.302585092994046)
(4.605170185988092)
(1.1631508247068418)
(1.722766580711205)
(1.5040773967762742)
grunt>
```

**Apache Pig ROUND Function:** The Apache Pig ROUND function is used to round up the provided value to an integer. If the return type is float, the value rounded to an integer. However, if the return type is double, the value rounded to a long.

### Syntax

ROUND(expression)

### Example of ROUND Function

In this example, we return the round up value of the given expression.

### Steps to execute ROUND Function

- Create a text file in your local machine and insert the values.  
\$ nano mathround.txt

```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: mathround.txt Modified
3.2
5.6
4.5
-2.3
-4.5
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the values inserted in the text files.  
\$ cat mathround.txt

```
codegyani@ubuntu64server: ~$ nano mathround.txt
codegyani@ubuntu64server:~$ cat mathround.txt
3.2
5.6
4.5
-2.3
-4.5
codegyani@ubuntu64server:~$
```

- Upload the text files on HDFS in the specific directory.  
\$ hdfs dfs -put mathround.txt /pigexample
- Open the pig MapReduce run mode.  
\$ pig
- Load the file that contains the data.  
grunt> A = LOAD '/pigexample/mathround.txt' AS (a1:float) ;
- Now, execute and verify the data.  
grunt> DUMP A;

```
codegyani@ubuntu64server: ~
aggregation.
2019-02-27 22:41:14,248 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 22:41:14,290 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 22:41:14,466 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 22:41:14,474 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(3.2)
(5.6)
(4.5)
(-2.3)
(-4.5)
grunt>
```

- Let's return the round up value.  

```
grunt> Result = FOREACH A GENERATE ROUND(a1);
```

```
grunt> DUMP Result;
```

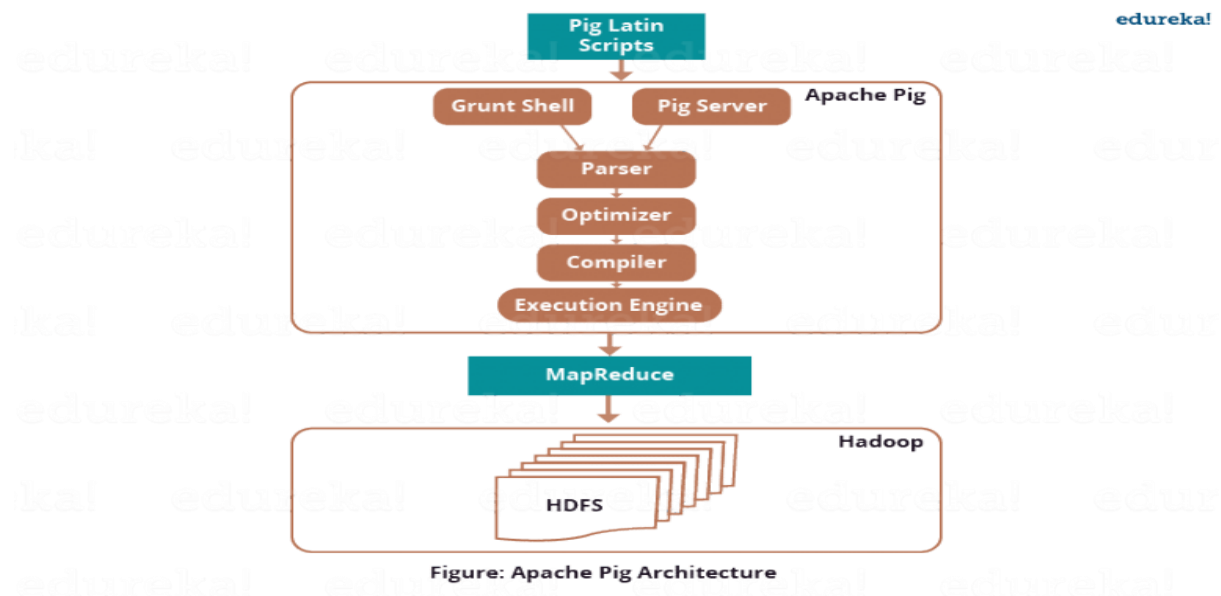
```

codegyani@ubuntu64server: ~
aggregation.
2019-02-27 22:49:20,793 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 22:49:20,798 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 22:49:20,833 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 22:49:20,837 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(3)
(6)
(5)
(-2)
(-4)
grunt>

```

## 8. Explain the PIG architecture and it's components.

Ans:



For writing a Pig script, we need Pig Latin language and to execute them, we need an execution environment. The architecture of Apache Pig is shown in the above image.

### Pig Latin Scripts

Initially as illustrated in the above image, we submit Pig scripts to the Apache Pig execution environment which can be written in Pig Latin using built-in operators.

There are three ways to execute the Pig script:

- Grunt Shell:** This is Pig's interactive shell provided to execute all Pig Scripts.
- Script File:** Write all the Pig commands in a script file and execute the Pig script file. This is executed by the Pig Server.
- Embedded Script:** If some functions are unavailable in built-in operators, we can programmatically create User Defined Functions to bring that functionalities using other languages like Java, Python, Ruby, etc. and embed it in Pig Latin Script file. Then, execute that script file.

### Parser

From the above image you can see, after passing through Grunt or Pig Server, Pig Scripts are passed to the Parser. The Parser does type checking and checks the syntax of the script. The

parser outputs a DAG (directed acyclic graph). DAG represents the Pig Latin statements and logical operators. The logical operators are represented as the nodes and the data flows are represented as edges.

### **Optimizer**

Then the DAG is submitted to the optimizer. The Optimizer performs the optimization activities like split, merge, transform, and reorder operators etc. This optimizer provides the automatic optimization feature to Apache Pig. The optimizer basically aims to reduce the amount of data in the pipeline at any instance of time while processing the extracted data

### **Compiler**

After the optimization process, the compiler compiles the optimized code into a series of MapReduce jobs. The compiler is the one who is responsible for converting Pig jobs automatically into MapReduce jobs.

### **Execution engine**

Finally, as shown in the figure, these MapReduce jobs are submitted for execution to the execution engine. Then the MapReduce jobs are executed and gives the required result. The result can be displayed on the screen using “**DUMP**” statement and can be stored in the HDFS using “**STORE**” statement.

## **9. Write and run the word count PIG script.**

**Ans:**

### **1)create a sample file for input:**

```
cat >> sample
```

```
This is a hadoop post
```

```
hadoop is a bigdata technology
```

### **2)upload it into HDFS:**

```
hadoop fs -put sample /pigexample/sample
```

### **3)open pig mapreduce/local mode:**

```
pig
```

### **4)load data:**

```
lines = LOAD '/PigExample/sample' AS (line:chararray);
```

The data we have is in sentences. So we have to convert that data into words using TOKENIZE Function.

```
(TOKENIZE(line));
```

(or)

If we have any delimiter like space we can specify as

```
(TOKENIZE(line,' '));
```

Output will be like this:

```
((This),(is),(a),(hadoop),(class))
```

```
((hadoop),(is),(a),(bigdata),(technology))
```

but we have to convert it into multiple rows like below

(This)  
(is)  
(a)  
(hadoop)  
(class)  
(hadoop)  
(is)  
(a)  
(bigdata)  
(technology)

5)words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;

6)grouped = GROUP words BY word;

(a,{(a),(a)})  
(is,{(is),(is)})  
(This,{(This)})  
(class,{(class)})  
(hadoop,{(hadoop),(hadoop)})  
(bigdata,{(bigdata)})  
(technology,{(technology)})

7)wordcount = FOREACH grouped GENERATE group, COUNT(words);

8)DUMP wordcount;

(a,2)  
(is,2)  
(This,1)  
(class,1)  
(hadoop,2)  
(bigdata,1)  
(technology,1)

## 10. Differentiate between PIG and SQL.

Ans:

| Pig                                                                                                                           | SQL                                                      |
|-------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| Pig Latin is a <b>procedural</b> language.                                                                                    | SQL is a <b>declarative</b> language.                    |
| In Apache Pig, <b>schema</b> is optional. We can store data without designing a schema (values are stored as \$01, \$02 etc.) | Schema is mandatory in SQL.                              |
| The data model in Apache Pig is <b>nested relational</b> .                                                                    | The data model used in SQL is <b>flat relational</b> .   |
| Apache Pig provides limited opportunity for <b>Query optimization</b> .                                                       | There is more opportunity for query optimization in SQL. |

In addition to above differences, Apache Pig Latin –



- Allows splits in the pipeline.
- Allows developers to store data anywhere in the pipeline.
- Declares execution plans.
- Provides operators to perform ETL (Extract, Transform, and Load) functions.

## 11. Explain the following data processing operators with examples.

### i. FOREACH      ii. GROUP   iii. LIMIT   iv. ORDERBY

Ans: Apache Pig FOREACH Operator: The Apache Pig FOREACH operator generates data transformations based on columns of data. It is recommended to use FILTER operation to work with tuples of data.

Example of FOREACH Operator

In this example, we traverse the data of two columns exists in the given file.

Steps to execute FOREACH Operator

- Create a text file in your local machine and provide some values to it.

\$ nano pforeach.txt

```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: pforeach.txt Modified
1,2,3
4,5,6
7,8,9
[New File]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the values written in the text files.

\$ cat pforeach.txt

```
codegyani@ubuntu64server: ~/PigExamples
codegyani@ubuntu64server:~/PigExamples$ nano pforeach.txt
codegyani@ubuntu64server:~/PigExamples$ cat pforeach.txt
1,2,3
4,5,6
7,8,9
codegyani@ubuntu64server:~/PigExamples$
```

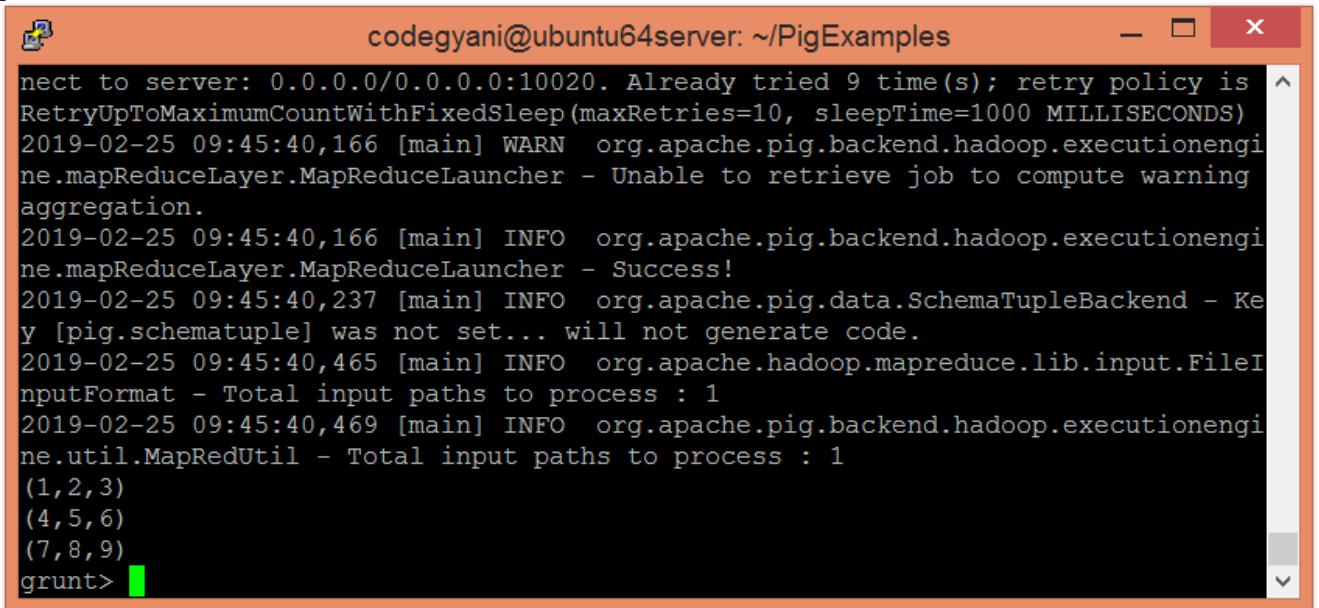
- Upload the text files on HDFS in the specific directory.

\$ hdfs dfs -put pforeach.txt /pigexample

- Open the pig MapReduce run mode.

\$ pig

- Load the file that contains the data.
- ```
grunt> A = LOAD '/pigexample/pforeach.txt' USING PigStorage(',') AS (a1:int,a2:int,a3:int);
```
- Now, execute and verify the data.
- ```
grunt> DUMP A;
```



```
codegyani@ubuntu64server: ~/PigExamples
nect to server: 0.0.0.0/0.0.0.0:10020. Already tried 9 time(s); retry policy is
RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-25 09:45:40,166 [main] WARN org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-25 09:45:40,166 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 09:45:40,237 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-25 09:45:40,465 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-25 09:45:40,469 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(1,2,3)
(4,5,6)
(7,8,9)
grunt>
```

- Let's traverse the data of two columns.
- ```
grunt> fe = FOREACH A generate a1,a2;
```
- ```
grunt> DUMP fe;
```



```
codegyani@ubuntu64server: ~/PigExamples
RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-25 09:55:54,433 [main] WARN org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-25 09:55:54,440 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 09:55:54,447 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-25 09:55:54,506 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-25 09:55:54,507 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(1,2)
(4,5)
(7,8)
grunt>
```

**Apache Pig Group Operator:** The Apache Pig GROUP operator is used to group the data in one or more relations. It groups the tuples that contain a similar group key. If the group key has more than one field, it treats as tuple otherwise it will be the same type as that of the group key. In a result, it provides a relation that contains one tuple per group.

Example of Group Operator

In this example, we group the given data on the basis of the last name.

Steps to execute Group Operator

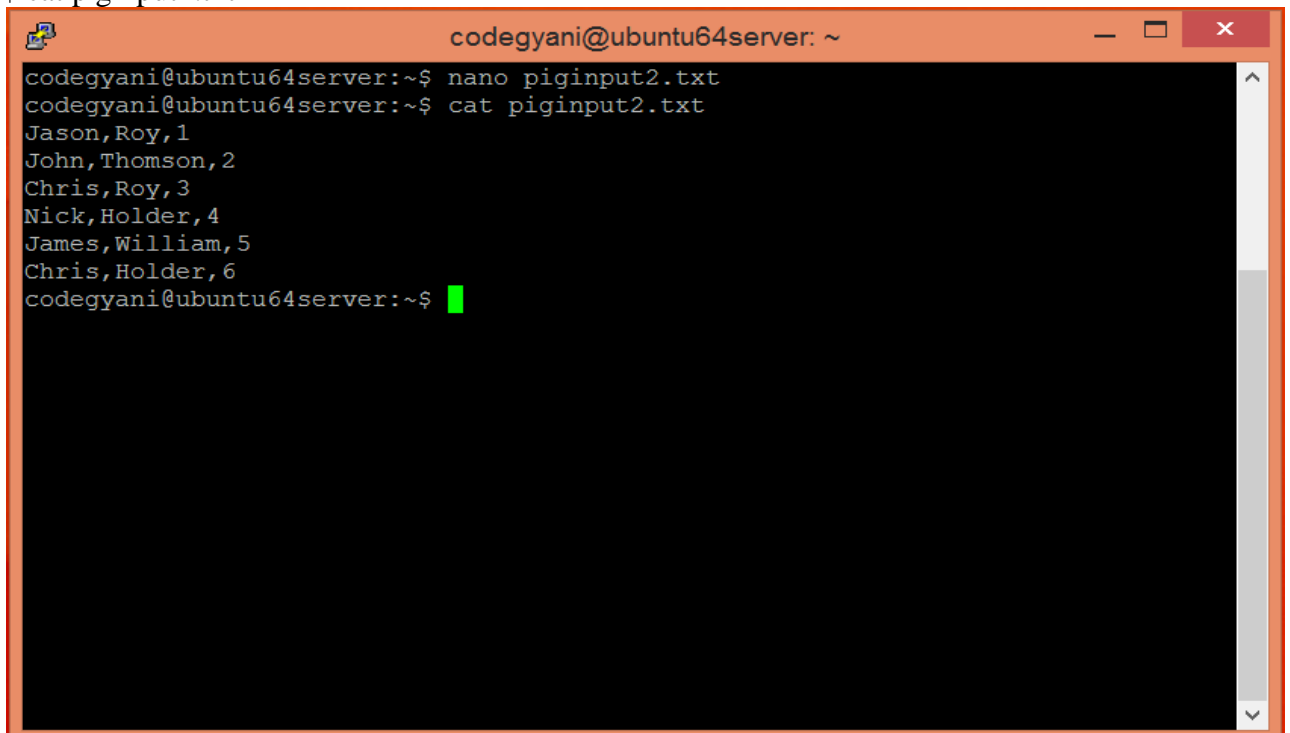
- Create a text file in your local machine and write some text into it.
- ```
$ nano piginput2.txt
```



```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: piginput2.txt Modified
Jason,Roy,1
John,Thomson,2
Chris,Roy,3
Nick,Holder,4
James,William,5
Chris,Holder,6
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the text written in the piginput2.txt file.

\$ cat piginput2.txt



```
codegyani@ubuntu64server: ~$ nano piginput2.txt
codegyani@ubuntu64server: ~$ cat piginput2.txt
Jason,Roy,1
John,Thomson,2
Chris,Roy,3
Nick,Holder,4
James,William,5
Chris,Holder,6
codegyani@ubuntu64server: ~$
```

- Upload the piginput2.txt file on HDFS in the specific directory.

\$ hdfs dfs -put /home/codegyani/piginput2.txt /pigexample

Browse Directory

/pigexample

Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	codegyani	supergroup	31 B	2/25/2019, 2:10:07 AM	1	128 MB	piginput1.txt
-rw-r--r--	codegyani	supergroup	84 B	2/25/2019, 3:20:21 AM	1	128 MB	piginput2.txt

Hadoop, 2015.

- Open the pig MapReduce run mode.

\$ pig

- Load the data into the bag.

```
grunt> A = LOAD '/pigexample/piginput2.txt' USING PigStorage(',') AS (fname:chararray,l_name:chararray,id:int);
```

- Now execute and verify the data.

```
grunt> DUMP A;
```

```
codegyani@ubuntu64server: ~
2019-02-24 17:03:17,151 [main] INFO org.apache.hadoop.ipc.Client - Retrying c
is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECO
2019-02-24 17:03:17,257 [main] WARN org.apache.pig.backend.hadoop.executionen
ng aggregation.
2019-02-24 17:03:17,258 [main] INFO org.apache.pig.backend.hadoop.executionen
2019-02-24 17:03:17,296 [main] INFO org.apache.pig.data.SchemaTupleBackend -
2019-02-24 17:03:17,587 [main] INFO org.apache.hadoop.mapreduce.lib.input.Fil
2019-02-24 17:03:17,594 [main] INFO org.apache.pig.backend.hadoop.executionen
(Jason,Roy,1)
(John,Thomson,2)
(Chris,Roy,3)
(Nick,Holder,4)
(James,William,5)
(Chris,Holder,6)
grunt>
grunt>
```

- Let us group the data on the basis of l_name.

```
grunt> groupgroupbyname = group A by l_name ;
```

- Now, execute and verify the data.

```
grunt> DUMP groupbyname;
```

```
codegyani@ubuntu64server: ~
is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECO
2019-02-24 17:17:29,102 [main] INFO org.apache.hadoop.ipc.Client - Retrying c
is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECO
2019-02-24 17:17:30,106 [main] INFO org.apache.hadoop.ipc.Client - Retrying c
is RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECO
2019-02-24 17:17:30,212 [main] WARN org.apache.pig.backend.hadoop.executionen
ng aggregation.
2019-02-24 17:17:30,213 [main] INFO org.apache.pig.backend.hadoop.executionen
2019-02-24 17:17:30,217 [main] INFO org.apache.pig.data.SchemaTupleBackend -
2019-02-24 17:17:30,276 [main] INFO org.apache.hadoop.mapreduce.lib.input.Fil
2019-02-24 17:17:30,281 [main] INFO org.apache.pig.backend.hadoop.executionen
(Roy, {(Chris,Roy,3),(Jason,Roy,1)})
(Holder, {(Chris,Holder,6),(Nick,Holder,4)})
(Thomson, {(John,Thomson,2)})
(William, {(James,William,5)})
grunt>
```

Apache Pig LIMIT Operator: The Apache Pig LIMIT operator is used to limit the number of output tuples. However, if you specify the limit of output tuples equal to or more than the number of tuples exists, all the tuples in the relation are returned.

Example of LIMIT Operator

In this example, we return only two tuples from all the tuples in the relation.

Steps to execute LIMIT Operator

- Create a text file in your local machine and insert the list of tuples.

\$ nano plimit.txt

```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: plimit.txt Modified
5,2,1
3,2,7
8,2,3
4,3,2
9,2,1
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the tuples inserted in the text files.

\$ cat plimit.txt

```
codegyani@ubuntu64server: ~/PigExamples
codegyani@ubuntu64server:~/PigExamples$ nano plimit.txt
codegyani@ubuntu64server:~/PigExamples$ cat plimit.txt
5,2,1
3,2,7
8,2,3
4,3,2
9,2,1
codegyani@ubuntu64server:~/PigExamples$
```

- Upload the text files on HDFS in the specific directory.
- ```
$ hdfs dfs -put plimit.txt /pigexample
```
- Open the pig MapReduce run mode.
- ```
$ pig
```
- Load the file that contains the data.
- ```
grunt> A = LOAD '/pigexample/plimit.txt' USING PigStorage(',') AS (a1:int,a2:int,a3:int) ;
```
- Now, execute and verify the data.
- ```
grunt> DUMP A;
```

```
codegyani@ubuntu64server: ~/PigExamples
2019-02-25 13:29:08,383 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-25 13:29:08,388 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 13:29:08,421 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-25 13:29:08,653 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 13:29:08,660 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(5,2,1)
(3,2,7)
(8,2,3)
(4,3,2)
(9,2,1)
grunt>
```

- Let's return the first two tuples.
- ```
grunt> Result = LIMIT A 2;
grunt> DUMP Result;
```

```
codegyani@ubuntu64server: ~/PigExamples
2019-02-25 13:32:45,887 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 13:32:45,892 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
2019-02-25 13:32:46,744 [main] INFO org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter - Saved output of task 'attempt_0001_m_000001_1' to hdfs://192.168.56.123:8020/tmp/temp-462791126/tmp-1745148875/_temporary/0/task_0001_m_000001
2019-02-25 13:32:46,974 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2019-02-25 13:32:46,992 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 13:32:46,994 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(5,2,1)
(3,2,7)
grunt>
```

**Apache Pig ORDER BY Operator:** The Apache Pig ORDER BY operator sorts a relation based on one or more fields. It maintains the order of tuples.

Example of ORDER BY Operator

In this example, we return only two tuples from all the tuples in the relation.

Steps to execute ORDER BY Operator

- Create a text file in your local machine and insert the list of tuples.

\$ nano porder.txt

```
codegyani@ubuntu64server: ~/PigExamples
GNU nano 2.2.6 File: porder.txt Modified
5,2,3
1,3,8
9,3,6
6,4,8
1,2,5
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the tuples inserted in the text files.

\$ cat porder.txt

```
codegyani@ubuntu64server: ~/PigExamples
codegyani@ubuntu64server:~/PigExamples$ nano porder.txt
codegyani@ubuntu64server:~/PigExamples$ cat porder.txt
5,2,3
1,3,8
9,3,6
6,4,8
1,2,5
codegyani@ubuntu64server:~/PigExamples$
```

- Upload the text files on HDFS in the specific directory.
- ```
$ hdfs dfs -put porder.txt /pigexample
```
- Open the pig MapReduce run mode.
- ```
$ pig
```
- Load the file that contains the data.
- ```
grunt> A = LOAD '/pigexample/porder.txt' USING PigStorage(',') AS (a1:int,a2:int,a3:int);
```
- Now, execute and verify the data.
- ```
grunt> DUMP A;
```

```
codegyani@ubuntu64server: ~/PigExamples
2019-02-25 14:38:38,872 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-25 14:38:38,874 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 14:38:38,903 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-25 14:38:39,062 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 14:38:39,068 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(5,2,3)
(1,3,8)
(9,3,6)
(6,4,8)
(1,2,5)
grunt>
```

- Let's return the first two tuples.
- ```
grunt> Result = ORDER A BY a1 DESC;
```
- ```
grunt> DUMP Result;
```



```
codegyani@ubuntu64server: ~/PigExamples
2019-02-25 14:57:12,386 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-25 14:57:12,391 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-25 14:57:12,420 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-25 14:57:12,496 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-25 14:57:12,497 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(9,3,6)
(6,4,8)
(5,2,3)
(1,2,5)
(1,3,8)
grunt>
```

## 12. Explain the following pre-defined PIG functions with examples.

**i.MAX                      ii.SIZE                      iii.SUM                      iv.TOKENIZE**

Ans: Apache Pig MAX Function: The Apache Pig MAX function is used to find out the maximum of the numeric values or chararrays in a single-column bag. It requires a preceding GROUP ALL statement for global maximums and a GROUP BY statement for group maximums. However, it ignores the NULL values.

### Syntax

MAX(exp)

Here,

**exp** - It is an expression with data types like chararray, int, float, long, etc.

### Example of MAX Function

In this example, we will find out the maximum of the given values.

### Steps to execute MAX Function

- Create a text file in your local machine and insert the list of tuples.  
\$ nano evalmax.txt

```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: evalavg.txt Modified
Rohan,a,2.4F
Rohan,b,2.8F
Rohan,c,3.6F
Sam,d,2.2F
Sam,e,3.4F
Sam,f,3.8F
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the tuples inserted in the text files.  
\$ cat evalmax.txt

```
codegyani@ubuntu64server: ~
codegyani@ubuntu64server:~$ cat evalmax.txt
Rohan,a,2.4F
Rohan,b,2.8F
Rohan,c,3.6F
Sam,d,2.2F
Sam,e,3.4F
Sam,f,3.8F
codegyani@ubuntu64server:~$
```

- Upload the text files on HDFS in the specific directory.  
\$ hdfs dfs -put evalmax.txt /pigexample
- Open the pig MapReduce run mode.  
\$ pig
- Load the file that contains the data.  
grunt> A = LOAD '/pigexample/evalmax.txt' USING PigStorage(',') AS (a1:chararray,a2:chararray,a3:float) ;
- Now, execute and verify the data.  
grunt> DUMP A;

```
codegyani@ubuntu64server: ~
aggregation.
2019-02-26 13:31:38,685 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 13:31:38,789 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 13:31:39,011 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 13:31:39,018 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Rohan,a,2.4)
(Rohan,b,2.8)
(Rohan,c,3.6)
(Sam,d,2.2)
(Sam,e,3.4)
(Sam,f,3.8)
grunt>
```

- Let's group the data on the basis of 'a1' field.  
grunt> B = GROUP A BY a1;  
grunt> DUMP B;

```
codegyani@ubuntu64server: ~
2019-02-26 13:42:35,637 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 13:42:35,674 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 13:42:35,798 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 13:42:35,800 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Sam, { (Sam, f, 3.8), (Sam, e, 3.4), (Sam, d, 2.2) })
(Rohan, { (Rohan, c, 3.6), (Rohan, b, 2.8), (Rohan, a, 2.4) })
grunt>
```

- Let's return the maximum of given numeric values.  
grunt> Result = FOREACH B GENERATE group, MAX(A.a3);  
grunt> DUMP Result;

```
codegyani@ubuntu64server: ~
2019-02-26 13:50:52,481 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 13:50:52,505 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 13:50:52,546 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 13:50:52,550 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Sam, 3.8)
(Rohan, 3.6)
grunt>
```

**Apache Pig SIZE Function:** The Apache Pig SIZE function is used to find the number of elements based on any Pig data type. It includes NULL values in size computation. Here, the size is not algebraic.

### Example of SIZE Function

In this example, we compute the number of characters present in the first field of each tuple.

#### Steps to execute SIZE Function

- Create a text file in your local machine and insert the list of tuples.  
\$ nano evalsize.txt

```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: evalsize.txt Modified
Java,James Gosling
C,Dennis Ritchie
JavaScript,Brendan Eich
Python,Guido van Rossum
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the tuples inserted in the text files.  
\$ cat evalsize.txt

```
codegyani@ubuntu64server: ~
codegyani@ubuntu64server:~$ nano evalsize.txt
codegyani@ubuntu64server:~$ cat evalsize.txt
Java,James Gosling
C,Dennis Ritchie
JavaScript,Brendan Eich
Python,Guido van Rossum
codegyani@ubuntu64server:~$
```

- Upload the text files on HDFS in the specific directory.  
\$ hdfs dfs -put evalsize.txt /pigexample
- Open the pig MapReduce run mode.  
\$ pig
- Load the file that contains the data.  
grunt> A = LOAD '/pigexample/evalsize.txt' USING PigStorage(',') AS (a1:chararray,a2:chararray);
- Now, execute and verify the data.  
grunt> DUMP A;

```
codegyani@ubuntu64server: ~
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-26 14:43:04,267 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 14:43:04,370 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 14:43:04,664 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 14:43:04,671 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Java,James Gosling)
(C,Dennis Ritchie)
(Javascript,Brendan Eich)
(Python,Guido van Rossum)
grunt>
```

- Let's return the size of the first field of each tuple.  
grunt> Result = FOREACH A GENERATE SIZE(a1);  
grunt> DUMP Result;

```
codegyani@ubuntu64server: ~
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-26 14:51:05,348 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 14:51:05,357 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-26 14:51:05,513 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-26 14:51:05,515 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(4)
(1)
(10)
(6)
grunt>
```

**Apache Pig SUM Function:** The Apache Pig SUM function is used to find the sum of the numeric values in a single-column bag. It requires a preceding GROUP ALL statement for global sums and a GROUP BY statement for group sums. It ignores the null values.

### Example of SUM Function

In this example, we compute the sum of given numeric values.

#### Steps to execute SUM Function

- Create a text file in your local machine and insert the list of tuples.  
\$ nano evalsum.txt

```
codegyani@ubuntu64server: ~
GNU nano 2.2.6 File: evalsum.txt Modified
Rohan,a,2.4F
Rohan,b,2.8F
Rohan,c,3.6F
Sam,d,2.2F
Sam,e,3.4F
Sam,f,3.8F
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

- Check the tuples inserted in the text files.  
\$ cat evalsum.txt

```
codegyani@ubuntu64server: ~$ nano evalsum.txt
codegyani@ubuntu64server:~$ cat evalsum.txt
Rohan,a,2.4F
Rohan,b,2.8F
Rohan,c,3.6F
Sam,d,2.2F
Sam,e,3.4F
Sam,f,3.8F
codegyani@ubuntu64server:~$
```

- Upload the text files on HDFS in the specific directory.  
\$ hdfs dfs -put evalsum.txt /pigexample

- Open the pig MapReduce run mode.

\$ pig

- Load the file that contains the data.  
 grunt> A = LOAD '/pigexample/evalsum.txt' USING PigStorage(',') AS (a1:chararray, a2:chararray,a3:float) ;
- Now, execute and verify the data.  
 grunt> DUMP A;

```
codegyani@ubuntu64server: ~
2019-02-26 15:23:15,809 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 15:23:15,839 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 15:23:16,001 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 15:23:16,004 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Rohan,a,2.4)
(Rohan,b,2.8)
(Rohan,c,3.6)
(Sam,d,2.2)
(Sam,e,3.4)
(Sam,f,3.8)
grunt>
```

- Let's group the data on the basis of 'a1' field.

grunt> B = GROUP A BY a1;  
 grunt> DUMP B;

```
codegyani@ubuntu64server: ~
RetryUpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
2019-02-26 15:30:31,694 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-26 15:30:31,698 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 15:30:31,706 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 15:30:31,746 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 15:30:31,748 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Sam,{(Sam,f,3.8),(Sam,e,3.4),(Sam,d,2.2)})
(Rohan,{(Rohan,c,3.6),(Rohan,b,2.8),(Rohan,a,2.4)})
grunt>
```

- Let's return the sum of given numeric values.

grunt> Result = FOREACH B GENERATE group, SUM(A.a3);  
 grunt> DUMP Result;

```
codegyani@ubuntu64server: ~
2019-02-26 15:39:41,216 [main] WARN org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning aggregation.
2019-02-26 15:39:41,220 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-02-26 15:39:41,260 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.
2019-02-26 15:39:41,342 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1
2019-02-26 15:39:41,344 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Sam,9.400000095367432)
(Rohan,8.799999952316284)
grunt>
```

**Apache Pig TOKENIZE Function:**The Apache Pig TOKENIZE function is used to splits the existing string and generates a bag of words in a result.

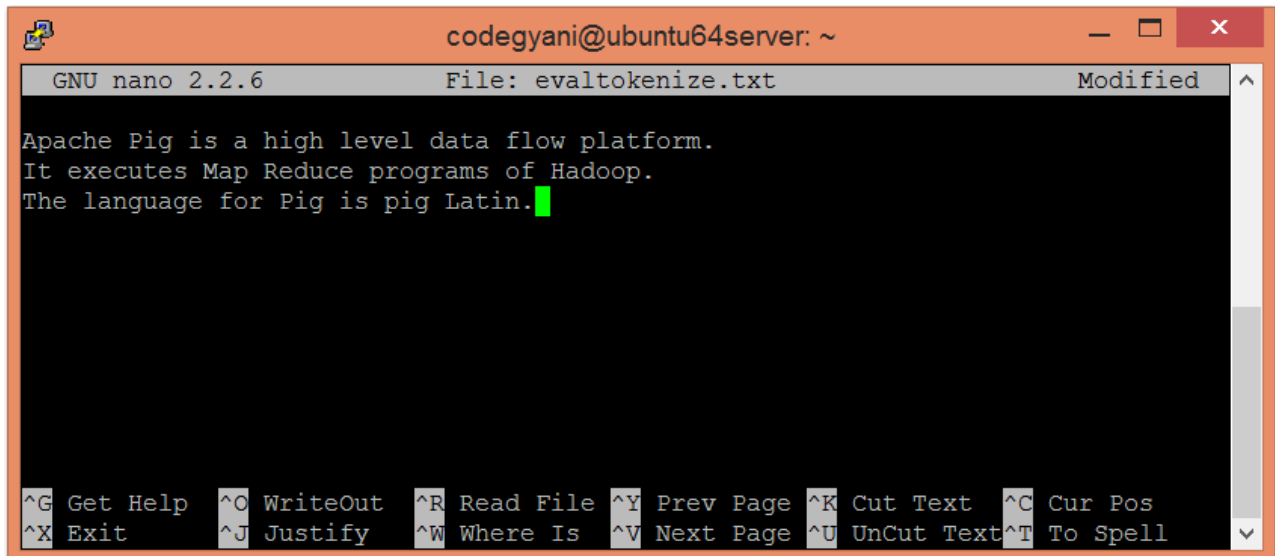
### Example of TOKENIZE Function

In this example, we split the string in the tokens.

### Steps to execute TOKENIZE Function

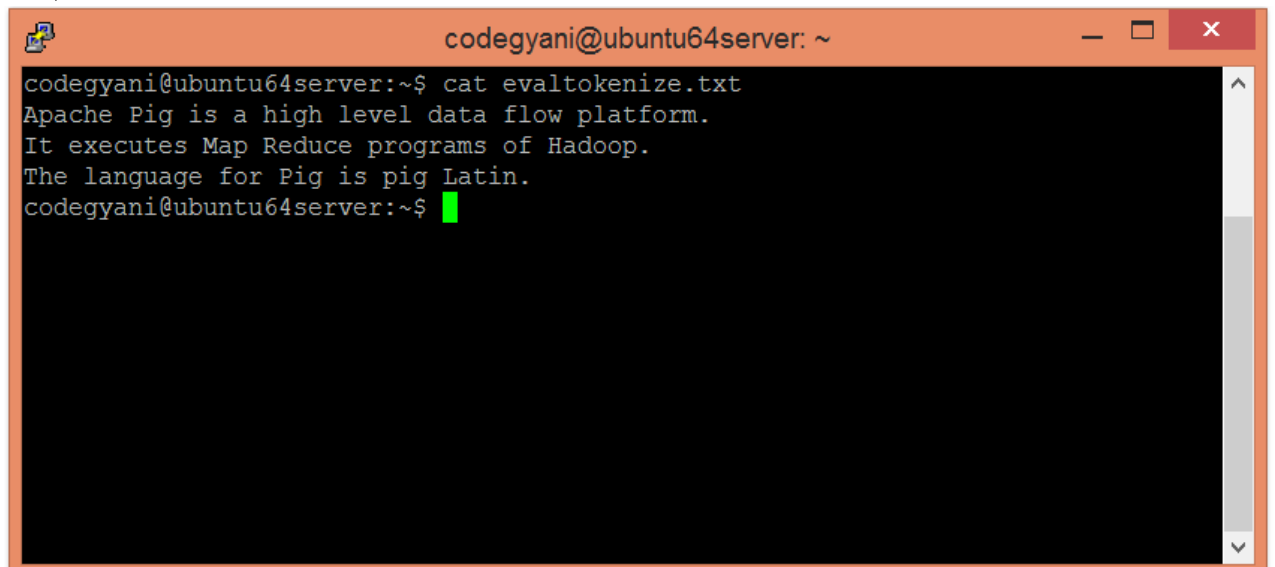
- Create a text file in your local machine and insert the list of tuples.

\$ nano evaltokenize.txt

A screenshot of a terminal window with a nano text editor. The window title is 'codegyani@ubuntu64server: ~'. The nano header shows 'GNU nano 2.2.6', 'File: evaltokenize.txt', and 'Modified'. The text inside the editor is: 'Apache Pig is a high level data flow platform.', 'It executes Map Reduce programs of Hadoop.', and 'The language for Pig is pig Latin.' followed by a green cursor. The bottom status bar shows various keyboard shortcuts like '^G Get Help', '^O WriteOut', '^R Read File', etc.

- Check the tuples inserted in the text files.

\$ cat evaltokenize.txt

A screenshot of a terminal window with the title 'codegyani@ubuntu64server: ~'. It shows the command 'codegyani@ubuntu64server:~\$ cat evaltokenize.txt' and its output: 'Apache Pig is a high level data flow platform.', 'It executes Map Reduce programs of Hadoop.', and 'The language for Pig is pig Latin.' followed by a green cursor on the next line.

- Upload the text files on HDFS in the specific directory.  
\$ hdfs dfs -put evaltokenize.txt /pigexample
- Open the pig MapReduce run mode.  
\$ pig
- Load the file that contains the data.  
grunt> A = LOAD '/pigexample/evaltokenize.txt' USING PigStorage(',') AS (a1:chararray) ;
- Now, execute and verify the data.  
grunt> DUMP A;



```

codegyani@ubuntu64server: ~
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-27 01:29:01,728 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 01:29:01,759 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 01:29:01,924 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 01:29:01,930 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(Apache Pig is a high level data flow platform.)
(It executes Map Reduce programs of Hadoop.)
(The language for Pig is pig Latin.)
grunt>

```

- Let's return the tokenized form of the string.  
 grunt> Result = FOREACH A GENERATE TOKENIZE(a1);  
 grunt> DUMP Result;

```

codegyani@ubuntu64server: ~
ne.mapReduceLayer.MapReduceLauncher - Unable to retrieve job to compute warning
aggregation.
2019-02-27 01:35:43,994 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.mapReduceLayer.MapReduceLauncher - Success!
2019-02-27 01:35:44,000 [main] INFO org.apache.pig.data.SchemaTupleBackend - Ke
y [pig.schematuple] was not set... will not generate code.
2019-02-27 01:35:44,134 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileI
nputFormat - Total input paths to process : 1
2019-02-27 01:35:44,140 [main] INFO org.apache.pig.backend.hadoop.executionengi
ne.util.MapRedUtil - Total input paths to process : 1
(({(Apache),(Pig),(is),(a),(high),(level),(data),(flow),(platform.))})
(({(It),(executes),(Map),(Reduce),(programs),(of),(Hadoop.))})
(({(The),(language),(for),(Pig),(is),(pig),(Latin.))})
grunt>

```

**Syllabus part-3: Hive: Hive Shell, Hive Services, Hive Metastore, Comparison with Traditional Databases, HiveQL, Tables, Querying Data and User Defined Functions.**

**13. Differentiate between HIVE and SQL.**

Ans: Comparison Chart

| Feature     | SQL                                                             | HiveQL                                                                             |
|-------------|-----------------------------------------------------------------|------------------------------------------------------------------------------------|
| Updates     | UPDATE, DELETE<br>INSERT,                                       | UPDATE, DELETE<br>INSERT,                                                          |
| Transaction | Supported                                                       | Limited Support Supported                                                          |
| Indexes     | Supported                                                       | Supported                                                                          |
| Data Types  | Integral, floating-point, fixed-point, text and binary strings, | Boolean, integral, floating-point, fixed-point, text and binary strings, temporal, |



|                            |                                |                                                                                                  |
|----------------------------|--------------------------------|--------------------------------------------------------------------------------------------------|
|                            | temporal                       | array, map, struct                                                                               |
| Functions                  | Hundreds of built-in functions | Hundreds of built-in functions                                                                   |
| Multitable inserts         | Not supported                  | Supported                                                                                        |
| Create table.....as Select | Not supported                  | Supported                                                                                        |
| Select                     | Supported                      | Supported with SORT BY clause for partial ordering and LIMIT to restrict number of rows returned |
| Joins                      | Supported                      | Inner joins, outer joins, semi join, map joins, cross joins                                      |
| Subqueries                 | Used in any clause             | Used in FROM, WHERE, or HAVING clauses                                                           |
| Views                      | Updatable                      | Read-only                                                                                        |

#### 14. Differentiate between PIG and HIVE.

Ans:

Hive vs Pig

| <i>Pig vs Hive - Differences</i>              |                                                                                           |
|-----------------------------------------------|-------------------------------------------------------------------------------------------|
| <b>Pig</b>                                    | <b>Hive</b>                                                                               |
| Procedural Data Flow Language                 | Declarative SQLish Language                                                               |
| For Programming                               | For creating reports                                                                      |
| Mainly used by Researchers and Programmers    | Mainly used by Data Analysts                                                              |
| Operates on the client side of a cluster.     | Operates on the server side of a cluster.                                                 |
| Does not have a dedicated metadata database.  | Makes use of exact variation of dedicated SQL DDL language by defining tables beforehand. |
| Pig is SQL like but varies to a great extent. | Directly leverages SQL and is easy to learn for database experts.                         |
| Pig supports Avro file format.                | Hive does not support it.                                                                 |

#### 15. What is HIVE? Explain various features of HIVE.

Ans: Apache Hive is a data warehouse system built on top of Hadoop and is used for analyzing structured and semi-structured data. Hive abstracts the complexity of Hadoop MapReduce. Basically, it provides a mechanism to project structure onto the data and perform queries written in HQL (Hive Query Language) that are similar to SQL statements. Internally, these queries or HQL gets converted to map reduce jobs by the Hive compiler. Therefore, you don't need to worry about writing complex MapReduce programs to process your data using Hadoop. It is targeted towards users who are comfortable with SQL. Apache Hive supports Data Definition Language (DDL), Data Manipulation Language (DML) and User Defined Functions (UDF).

## Advantages of Hive

- Useful for people who aren't from a programming background as it eliminates the need to write complex MapReduce program.
- **Extensible** and **scalable** to cope up with the growing volume and variety of data, without affecting performance of the system.
- It is as an efficient ETL (Extract, Transform, Load) tool.
- Hive supports any client application written in Java, PHP, Python, C++ or Ruby by exposing its **Thrift server**. (You can use these client – side languages embedded with SQL for accessing a database such as DB2, etc.).
- As the metadata information of Hive is stored in an RDBMS, it significantly reduces the time to perform semantic checks during query execution.

## 16. Describe the HIVE architecture and it's components.

Ans: The following image describes the Hive Architecture and the flow in which a query is submitted into Hive and finally processed using the MapReduce framework:

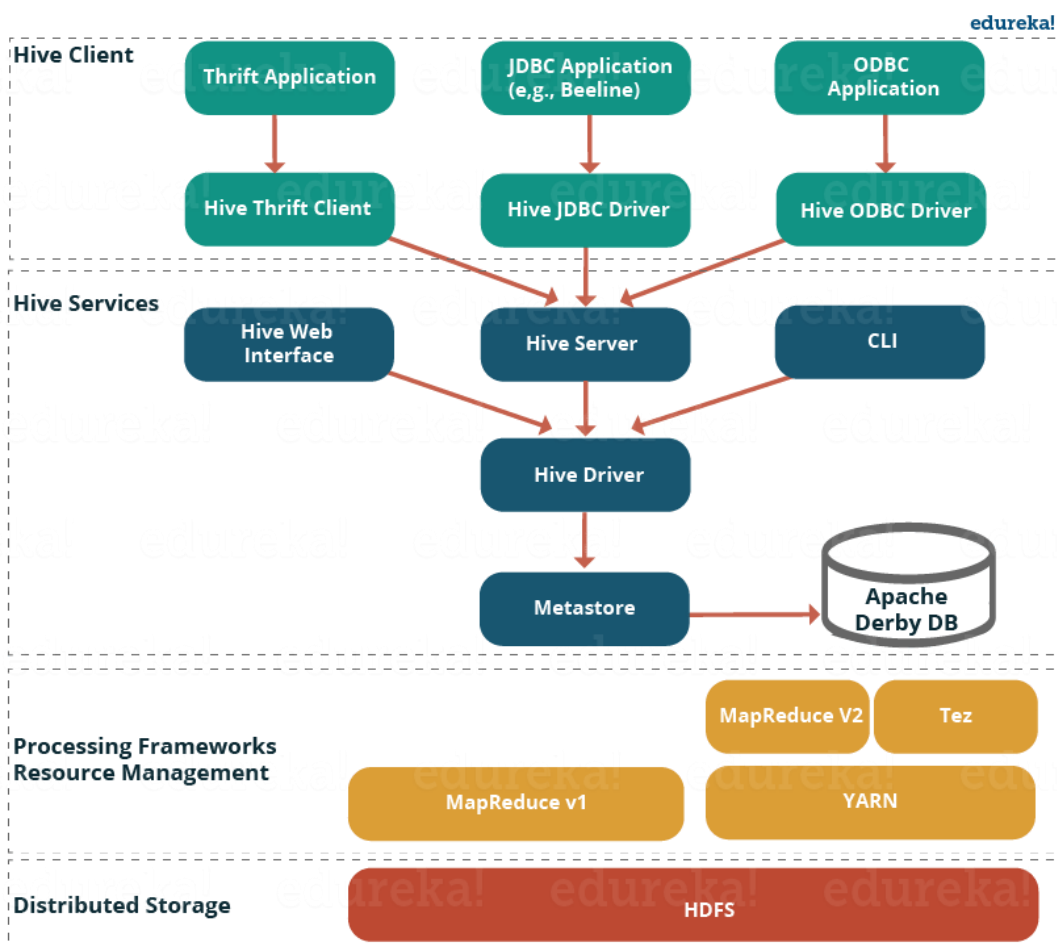


Fig: *Hive Architecture*

As shown in the above image, the Hive Architecture can be categorized into the following components:

- **Hive Clients:** Hive supports application written in many languages like Java, C++, Python etc. using JDBC, Thrift and ODBC drivers. Hence one can always write hive client application written in a language of their choice.
- **Hive Services:** Apache Hive provides various services like CLI, Web Interface etc. to perform queries.

- **Processing framework and Resource Management:** Internally, Hive uses Hadoop MapReduce framework as de facto engine to execute the queries.
- **Distributed Storage:** As Hive is installed on top of Hadoop, it uses the underlying HDFS for the distributed storage.

## 17. Explain the DDL commands in HIVE.

Ans: DDL Commands in Hive

SQL users might already be familiar with what DDL commands are but for readers who are new to SQL, DDL refers to Data Definition Language. DDL commands are the statements that are responsible for defining and changing the structure of a database or table in Hive.

| DDL Commands in Hive |                                                                   |
|----------------------|-------------------------------------------------------------------|
| CREATE               | Database, Table                                                   |
| DROP                 | Database, Table                                                   |
| TRUNCATE             | Table                                                             |
| ALTER                | Database, Table                                                   |
| SHOW                 | Databases, Tables, Table Properties, Partitions, Functions, Index |
| DESCRIBE             | Database, Table, View                                             |

Let's look at the usage of the top hive commands in HQL on both databases and tables –

### DDL Commands on Databases in Hive

#### Create Database in Hive

As the name implies, this DDL command in Hive is used for creating databases.

**CREATE (DATABASE) [IF NOT EXISTS] database\_name**

**[COMMENT database\_comment]**

**[LOCATION hdfs\_path]**

**[WITH DBPROPERTIES (property\_name=property\_value, ...)];**

In the above syntax for create database command, the values mentioned in square brackets [] are optional.

#### *Usage of Create Database Command in Hive*

```
hive> create database if not exists firstDB comment "This is my first demo" location
'/user/hive/warehouse/newdb' with DBPROPERTIES
('createdby'='abhay','createdfor'='dezyre');
OK
Time taken: 0.092 seconds
```

#### Drop Database in Hive

This command is used for deleting an already created database in Hive and the syntax is as follows -

**DROP (DATABASE) [IF EXISTS] database\_name [RESTRICT|CASCADE];**

### *Usage of Drop Database Command in Hive*

```
hive> drop database if exists firstDB CASCADE;
OK
Time taken: 0.099 seconds
```

In Hadoop Hive, the mode is set as RESTRICT by default and users cannot delete it unless it is non-empty. For deleting a database in Hive along with the existing tables, users must change the mode from RESTRICT to CASCADE.

In the syntax for drop database Hive command, “if exists” clause is used to avoid any errors that might occur if the programmer tries to delete a database which does not exist.

### *Describe Database Command in Hive*

This command is used to check any associated metadata for the databases.

```
hive> describe database extended firstDB;
OK
firstdb This is my first demo hdfs://Boss-Machine:9000/user/hive/warehouse/new
db abhay USER {createdby=abhay, createdfor=dezyre}
Time taken: 0.027 seconds, Fetched: 1 row(s)
```

### *Alter Database Command in Hive*

Whenever the developers need to change the metadata of any of the databases, alter hive DDL command can be used as follows –

```
ALTER (DATABASE) database_name SET DBPROPERTIES
(property_name=property_value, ...);
```

### **Usage of ALTER database command in Hive –**

Let’s use the Alter command to modify the OWNER property and specify the role for the owner –

```
ALTER (DATABASE) database_name SET OWNER [USER|ROLE] user_or_role;
```

```
hive> alter database firstDB set OWNER ROLE admin;
OK
Time taken: 0.058 seconds
hive> describe database extended firstDB;
OK
firstdb This is my first demo hdfs://Boss-Machine:9000/user/hive/warehouse/new
db admin ROLE {createdby=abhay, createdfor=dezyre}
Time taken: 0.05 seconds, Fetched: 1 row(s)
```

### *Show Database Command in Hive*

Programmers can view the list of existing databases in the current schema.

### *Usage of Show Database Command*

```
Show databases;
```

```
hive> Show databases;
OK
default
firstdb
Time taken: 0.039 seconds, Fetched: 2 row(s)
```

### Use Database Command in Hive

This hive command is used to select a specific database for the session on which hive queries would be executed.

#### *Usage of Use Database Command in Hive*

```
hive> use firstdb;
OK
Time taken: 0.027 seconds
```

### DDL Commands on Tables in Hive

#### *Create Table Command in Hive*

Hive create table command is used to create a table in the existing database that is in use for a particular session.

CREATE TABLE [IF NOT EXISTS] [db\_name.]table\_name --

[(col\_name data\_type [COMMENT col\_comment], ...)]

[COMMENT table\_comment]

[LOCATION hdfs\_path]

## Hive Create Table Usage

```
hive> create database IF NOT EXISTS college;
OK
Time taken: 0.062 seconds
hive> use college;
OK
Time taken: 0.021 seconds
hive>
 > CREATE TABLE IF NOT EXISTS college.students (
 > ID BIGINT COMMENT 'unique id for each student',
 > name STRING COMMENT 'student name',
 > age INT COMMENT 'student age between 16-26',
 > fee DOUBLE COMMENT 'student college fee',
 > city STRING COMMENT 'cities to which students belongs',
 > state STRING COMMENT 'student home address streets',
 > zip BIGINT COMMENT 'student address zip code'
 >)
 > COMMENT 'This table holds the demography info for each student'
 > ROW FORMAT DELIMITED
 > FIELDS TERMINATED BY '|'
 > LINES TERMINATED BY '\n'
 > STORED AS TEXTFILE
 > LOCATION '/user/hive/warehouse/college.db/students';
OK
Time taken: 0.317 seconds
```

In the above step, we have created a hive table named Students in the database college with various fields like ID, Name, fee, city, etc. Comments have been mentioned for each column so that anybody referring to the table gets an overview about what the columns mean.

The LOCATION keyword is used for specifying where the table should be stored on HDFS.

How to create a table in hive by copying an existing table schema?

Hive lets programmers create a new table by replicating the schema of an existing table but remember only the schema of the new table is replicated but not the data. When creating the new table, the location parameter can be specified.

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name Like
[db_name].existing_table [LOCATION hdfs_path]
```

```
hive> CREATE TABLE IF NOT EXISTS college.senior_students LIKE college.students
LOCATION '/user/hive/warehouse/college.db/senior_students';
OK
Time taken: 0.307 seconds
```

## DROP Table Command in Hive

Drops the table and all the data associated with it in the Hive metastore.

```
DROP TABLE [IF EXISTS] table_name [PURGE];
```

## Usage of DROP Table command in Hive

```
hive> drop table if exists senior_students PURGE;
OK
Time taken: 0.58 seconds
```

DROP table command removes the metadata and data for a particular table. Data is usually moved to .Trash/Current directory if Trash is configured. If PURGE option is specified then the table data will not go to the trash directory and there will be no scope to retrieve the data in case of erroneous DROP command execution.

### TRUNCATE Table Command in Hive

This hive command is used to truncate all the rows present in a table i.e. it deletes all the data from the Hive meta store and the data cannot be restored.

TRUNCATE TABLE [db\_name].table\_name

*Usage of TRUNCATE Table in Hive*

```
hive> truncate table college.senior_students;
OK
Time taken: 0.141 seconds
```

### ALTER Table Command in Hive

Using ALTER Table command, the structure and metadata of the table can be modified even after the table has been created. Let's try to change the name of an existing table using the ALTER command –

ALTER TABLE [db\_name].old\_table\_name RENAME TO [db\_name].new\_table\_name;

```
hive> ALTER TABLE college.senior_students RENAME TO college.college_students;
OK
Time taken: 0.247 seconds
```

### Syntax to ALTER Table Properties

ALTER TABLE [db\_name].tablename SET TBLPROPERTIES  
(‘property\_key’=‘property\_new\_value’)

```
hive> ALTER TABLE college.college_students SET TBLPROPERTIES ('creator' = 'abhay
<div data-bbox="114 710 868 744" data-label="Text">

In the above step, we have set the creator attribute for the table and similarly we can later or modify other table properties also.hive> ALTER TABLE college.college_students SET TBLPROPERTIES ('comment' = 'changed the creator to abhay');
OK
Time taken: 0.161 seconds


```

</div>
<div data-bbox="114 834 424 851" data-label="Section-Header">

### DESCRIBE Table Command in Hive

</div>
<div data-bbox="114 869 704 888" data-label="Text">

Gives the information of a particular table and the syntax is as follows –

</div>
<div data-bbox="114 886 776 920" data-label="Text">

DESCRIBE [EXTENDED|FORMATTED] [db\_name.] table\_name[.col\_name (

</div>
<div data-bbox="114 935 425 954" data-label="Section-Header">

### Usage of Describe Table Command

```
hive> describe college.college_students;
OK
id bigint unique id for each student
name string student name
age int student age between 16-26
fee double student college fee
city string cities to which students belongs
state string student home address streets
zip bigint student address zip code
Time taken: 0.064 seconds, Fetched: 7 row(s)
```

```
hive> describe extended college.college_students;
OK
id bigint unique id for each student
name string student name
age int student age between 16-26
fee double student college fee
city string cities to which students belongs
state string student home address state s
zip bigint student address zip code
```

We can also check the description for a specific column from the table as follows –

```
hive> describe extended college.college_students name;
OK
name string from deserializer
Time taken: 0.076 seconds, Fetched: 1 row(s)
```

Show Table Command in Hive

Gives the list of existing tables in the current database schema.

```
hive> show tables;
OK
college_students
students
Time taken: 0.018 seconds, Fetched: 2 row(s)
```

## 18. How many ways to create tables in HIVE explain each one with own example.

Ans: Tables in Hive are the same as the tables present in a Relational Database. You can perform filter, project, join and union operations on them. There are two types of tables in Hive:

1. *Managed(Internal) Table:*

*Command:*

*CREATE TABLE <table\_name> (column1 data\_type, column2 data\_type);*

*LOAD DATA INPATH <HDFS\_file\_location> INTO table managed\_table;*

As the name suggests (managed table), Hive is responsible for managing the data of a managed table. In other words, what I meant by saying, “Hive manages the data”, is that if you load the data from a file present in HDFS into a Hive *Managed Table* and issue a DROP



command on it, the table along with its metadata will be deleted. So, the data belonging to the dropped *managed\_table* no longer exist anywhere in HDFS and you can't retrieve it by any means. Basically, you are moving the data when you issue the LOAD command from the HDFS file location to the Hive warehouse directory.

**Note:** The default path of the warehouse directory is set to `/user/hive/warehouse`. The data of a Hive table resides in `warehouse_directory/table_name` (HDFS). You can also specify the path of the warehouse directory in the `hive.metastore.warehouse.dir` configuration parameter present in the `hive-site.xml`.

## 2. External Table:

Command:

```
CREATE EXTERNAL TABLE <table_name> (column1 data_type, column2 data_type)
LOCATION '<table_hive_location>';
```

```
LOAD DATA INPATH '<HDFS_file_location>' INTO TABLE <table_name>;
```

For *external table*, Hive is not responsible for managing the data. In this case, when you issue the LOAD command, Hive moves the data into its warehouse directory. Then, Hive creates the metadata information for the external table. Now, if you issue a DROP command on the *external table*, only metadata information regarding the external table will be deleted. Therefore, you can still retrieve the data of that very external table from the warehouse directory using HDFS commands.

## 7. What is table partition? Explain different table partitions.

Ans:

Partitions:

Command:

```
CREATE TABLE table_name (column1 data_type, column2 data_type) PARTITIONED BY
(partition1 data_type, partition2 data_type,...);
```

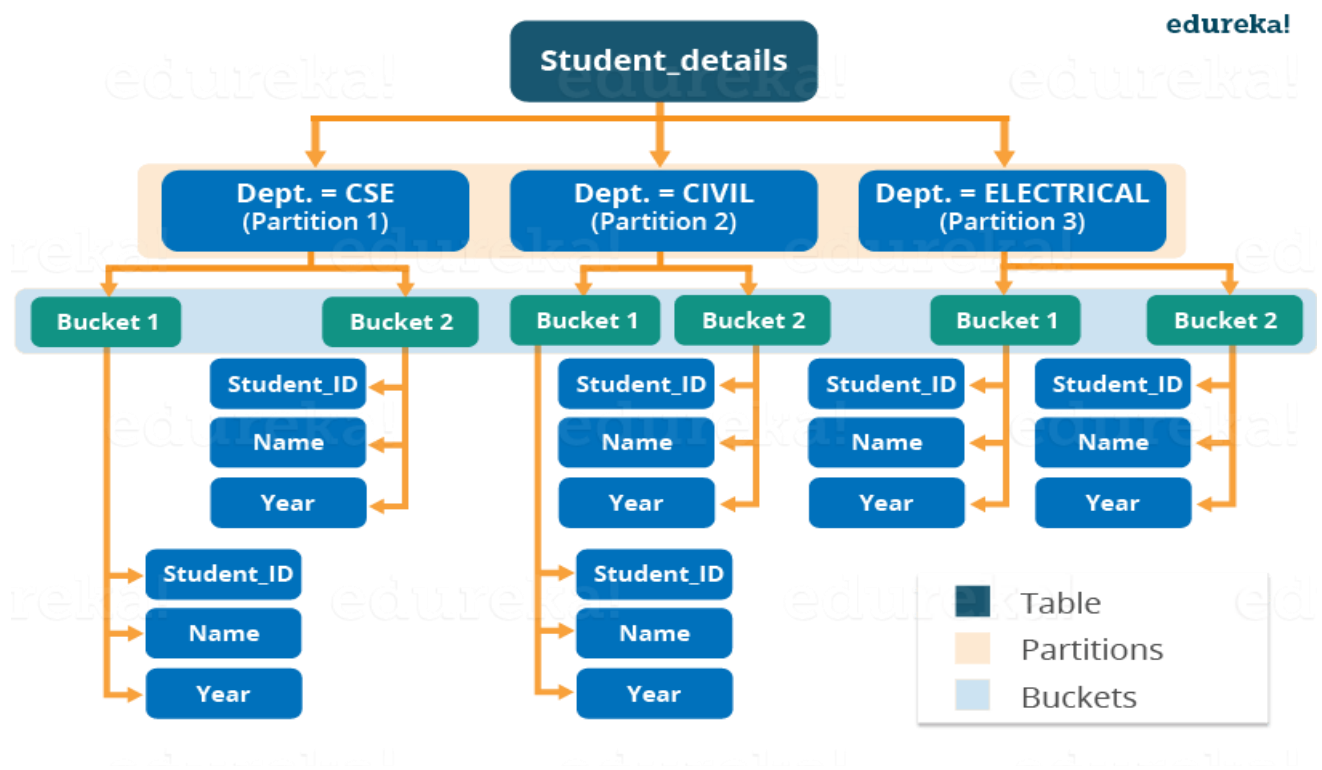
Hive organizes tables into partitions for grouping similar type of data together based on a column or partition key. Each Table can have one or more partition keys to identify a particular partition. This allows us to have a faster query on slices of the data.

**Note:** Remember, the most common mistake made while creating partitions is to specify an existing column name as a partition column. While doing so, you will receive an error – “Error in semantic analysis: Column repeated in partitioning columns”.

Let us understand partition by taking an example where I have a table `student_details` containing the student information of some engineering college like `student_id`, `name`, `department`, `year`, etc. Now, if I perform partitioning based on `department` column, the information of all the students belonging to a particular department will be stored together in that very partition. Physically, a partition is nothing but a sub-directory in the table directory.

Let's say we have data for three departments in our `student_details` table – CSE, ECE and Civil. Therefore, we will have three partitions in total for each of the departments as shown in the image below. And, for each department we will have all the data regarding that very department residing in a separate sub – directory under the Hive table directory. For example, all the student data regarding CSE departments will be stored in `user/hive/warehouse/student_details/dept.=CSE`. So, the queries regarding CSE students would only have to look through the data present in the CSE partition. This makes

partitioning very useful as it reduces the query latency by scanning only **relevant** partitioned data instead of the whole data set. In fact, in real world implementations, you will be dealing with hundreds of TBs of data. So, imagine scanning this huge amount of data for some query where **95%** data scanned by you was un-relevant to your query.



## 19. Describe the various pre-defined functions supported by HIVE.

Ans: Built-In Functions

Hive supports the following built-in functions:

| Return Type | Signature                               | Description                                                                          |
|-------------|-----------------------------------------|--------------------------------------------------------------------------------------|
| BIGINT      | round(double a)                         | It returns the rounded BIGINT value of the double.                                   |
| BIGINT      | floor(double a)                         | It returns the maximum BIGINT value that is equal or less than the double.           |
| BIGINT      | ceil(double a)                          | It returns the minimum BIGINT value that is equal or greater than the double.        |
| double      | rand(), rand(int seed)                  | It returns a random number that changes from row to row.                             |
| string      | concat(string A, string B,...)          | It returns the string resulting from concatenating B after A.                        |
| string      | substr(string A, int start)             | It returns the substring of A starting from start position till the end of string A. |
| string      | substr(string A, int start, int length) | It returns the substring of A starting from start position with the given length.    |
| string      | upper(string A)                         | It returns the string resulting from converting all characters of A to upper case.   |
| string      | ucase(string A)                         | Same as above.                                                                       |
| string      | lower(string A)                         | It returns the string resulting from converting                                      |

|                 |                                                  |                                                                                                                                                                                                      |
|-----------------|--------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 |                                                  | all characters of B to lower case.                                                                                                                                                                   |
| string          | lcase(string A)                                  | Same as above.                                                                                                                                                                                       |
| string          | trim(string A)                                   | It returns the string resulting from trimming spaces from both ends of A.                                                                                                                            |
| string          | ltrim(string A)                                  | It returns the string resulting from trimming spaces from the beginning (left hand side) of A.                                                                                                       |
| string          | rtrim(string A)                                  | rtrim(string A) It returns the string resulting from trimming spaces from the end (right hand side) of A.                                                                                            |
| string          | regexp_replace(string A, string B, string C)     | It returns the string resulting from replacing all substrings in B that match the Java regular expression syntax with C.                                                                             |
| int             | size(Map<K.V>)                                   | It returns the number of elements in the map type.                                                                                                                                                   |
| int             | size(Array<T>)                                   | It returns the number of elements in the array type.                                                                                                                                                 |
| value of <type> | cast(<expr> as <type>)                           | It converts the results of the expression expr to <type> e.g. cast('1' as BIGINT) converts the string '1' to it integral representation. A NULL is returned if the conversion does not succeed.      |
| string          | from_unixtime(int unixtime)                      | convert the number of seconds from Unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00" |
| string          | to_date(string timestamp)                        | It returns the date part of a timestamp string: to_date("1970-01-01 00:00:00") = "1970-01-01"                                                                                                        |
| int             | year(string date)                                | It returns the year part of a date or a timestamp string: year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970                                                                              |
| int             | month(string date)                               | It returns the month part of a date or a timestamp string: month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11                                                                               |
| int             | day(string date)                                 | It returns the day part of a date or a timestamp string: day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1                                                                                       |
| string          | get_json_object(string json_string, string path) | It extracts json object from a json string based on json path specified, and returns json string of the extracted json object. It returns NULL if the input json string is invalid.                  |

## 20. Describe the various operators supported by HIVE.

Ans: There are four types of operators in Hive:

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Complex Operators

## Relational Operators

These operators are used to compare two operands. The following table describes the relational operators available in Hive:

| Operator      | Operand             | Description                                                                                                 |
|---------------|---------------------|-------------------------------------------------------------------------------------------------------------|
| A = B         | all primitive types | TRUE if expression A is equivalent to expression B otherwise FALSE.                                         |
| A != B        | all primitive types | TRUE if expression A is not equivalent to expression B otherwise FALSE.                                     |
| A < B         | all primitive types | TRUE if expression A is less than expression B otherwise FALSE.                                             |
| A <= B        | all primitive types | TRUE if expression A is less than or equal to expression B otherwise FALSE.                                 |
| A > B         | all primitive types | TRUE if expression A is greater than expression B otherwise FALSE.                                          |
| A >= B        | all primitive types | TRUE if expression A is greater than or equal to expression B otherwise FALSE.                              |
| A IS NULL     | all types           | TRUE if expression A evaluates to NULL otherwise FALSE.                                                     |
| A IS NOT NULL | all types           | FALSE if expression A evaluates to NULL otherwise TRUE.                                                     |
| A LIKE B      | Strings             | TRUE if string pattern A matches to B otherwise FALSE.                                                      |
| A RLIKE B     | Strings             | NULL if A or B is NULL, TRUE if any substring of A matches the Java regular expression B , otherwise FALSE. |
| A REGEXP B    | Strings             | Same as RLIKE.                                                                                              |

## Example

Let us assume the **employee** table is composed of fields named Id, Name, Salary, Designation, and Dept as shown below. Generate a query to retrieve the employee details whose Id is 1205.

| Id   | Name        | Salary | Designation       | Dept  |
|------|-------------|--------|-------------------|-------|
| 1201 | Gopal       | 45000  | Technical manager | TP    |
| 1202 | Manisha     | 45000  | Proofreader       | PR    |
| 1203 | Masthanvali | 40000  | Technical writer  | TP    |
| 1204 | Krian       | 40000  | Hr Admin          | HR    |
| 1205 | Kranthi     | 30000  | Op Admin          | Admin |

The following query is executed to retrieve the employee details using the above table:  
hive> SELECT \* FROM employee WHERE Id=1205;

On successful execution of query, you get to see the following response:

| ID   | Name    | Salary | Designation | Dept  |
|------|---------|--------|-------------|-------|
| 1205 | Kranthi | 30000  | Op Admin    | Admin |

The following query is executed to retrieve the employee details whose salary is more than or equal to Rs 40000.

```
hive> SELECT * FROM employee WHERE Salary>=40000;
```

On successful execution of query, you get to see the following response:

| ID   | Name        | Salary | Designation       | Dept |
|------|-------------|--------|-------------------|------|
| 1201 | Gopal       | 45000  | Technical manager | TP   |
| 1202 | Manisha     | 45000  | Proofreader       | PR   |
| 1203 | Masthanvali | 40000  | Technical writer  | TP   |
| 1204 | Krian       | 40000  | Hr Admin          | HR   |

## Arithmetic Operators

These operators support various common arithmetic operations on the operands. All of them return number types. The following table describes the arithmetic operators available in Hive:

| Operators | Operand          | Description                                         |
|-----------|------------------|-----------------------------------------------------|
| A + B     | all number types | Gives the result of adding A and B.                 |
| A - B     | all number types | Gives the result of subtracting B from A.           |
| A * B     | all number types | Gives the result of multiplying A and B.            |
| A / B     | all number types | Gives the result of dividing B from A.              |
| A % B     | all number types | Gives the remainder resulting from dividing A by B. |
| A & B     | all number types | Gives the result of bitwise AND of A and B.         |
| A   B     | all number types | Gives the result of bitwise OR of A and B.          |
| A ^ B     | all number types | Gives the result of bitwise XOR of A and B.         |
| ~A        | all number types | Gives the result of bitwise NOT of A.               |

## Example

The following query adds two numbers, 20 and 30.

```
hive> SELECT 20+30 ADD FROM temp;
```

On successful execution of the query, you get to see the following response:

| ADD |
|-----|
| 50  |

## Logical Operators

The operators are logical expressions. All of them return either TRUE or FALSE.

| Operators | Operands | Description |
|-----------|----------|-------------|
|-----------|----------|-------------|

|         |         |                                                          |
|---------|---------|----------------------------------------------------------|
| A AND B | boolean | TRUE if both A and B are TRUE, otherwise FALSE.          |
| A && B  | boolean | Same as A AND B.                                         |
| A OR B  | boolean | TRUE if either A or B or both are TRUE, otherwise FALSE. |
| A    B  | boolean | Same as A OR B.                                          |
| NOT A   | boolean | TRUE if A is FALSE, otherwise FALSE.                     |
| !A      | boolean | Same as NOT A.                                           |

### Example

The following query is used to retrieve employee details whose Department is TP and Salary is more than Rs 40000.

```
hive> SELECT * FROM employee WHERE Salary>40000 && Dept=TP;
```

On successful execution of the query, you get to see the following response:

```
+-----+-----+-----+-----+-----+
| ID | Name | Salary | Designation | Dept |
+-----+-----+-----+-----+-----+
|1201 | Gopal | 45000 | Technical manager | TP |
+-----+-----+-----+-----+-----+
```

### Complex Operators

These operators provide an expression to access the elements of Complex Types.

| Operator | Operand                             | Description                                                               |
|----------|-------------------------------------|---------------------------------------------------------------------------|
| A[n]     | A is an Array and n is an int       | It returns the nth element in the array A. The first element has index 0. |
| M[key]   | M is a Map<K, V> and key has type K | It returns the value corresponding to the key in the map.                 |
| S.x      | S is a struct                       | It returns the x field of S.                                              |

## 21. Explain how do we create and access the HIVE user defined functions?

Ans: Create a Java class for the User Defined Function which extends `ora.apache.hadoop.hive.sql.exec.UDF` and implements more than one `evaluate()` methods. Put in your desired logic and you are almost there.

1. Package your Java class into a JAR file
2. Go to Hive CLI, add your JAR, and verify your JARs is in the Hive CLI classpath
3. CREATE TEMPORARY FUNCTION in Hive which points to your Java class
4. Use it in Hive SQL

There are better ways to do this, by writing your own `GenericUDF` to deal with non-primitive types like arrays and maps.

Create Java Class for a User Defined Function

As you can see below I am calling my Java class “Strip”. You can call it anything, but the important point is that it extends the `UDF` interface and provides two `evaluate()` implementations.

**evaluate(Text str, String stripChars)** - will trim specified characters in `stripChars` from first argument `str`.

**evaluate(Text str)** - will trim leading and trailing spaces.

```

package org.hardik.letsdobigdata;
import org.apache.commons.lang.StringUtils;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;
public class Strip extends UDF {
private Text result = new Text();
public Text evaluate(Text str, String stripChars) {
if(str == null) {
return null;
}
result.set(StringUtils.strip(str.toString(), stripChars));
return result;
}
public Text evaluate(Text str) {
if(str == null) {
return null;
}
result.set(StringUtils.strip(str.toString()));
return result;
}
}

```

Package Your Java Class into a JAR

```
$ cd HiveUDFs
```

and run "mvn clean package". This will create a JAR file which contains our UDF class.  
Copy the JAR's path.

Go to the Hive CLI and Add the UDF JAR

```
hive> ADD /home/cloudera/workspace/HiveUDFs/target/HiveUDFs-0.0.1-SNAPSHOT.jar;
Added [/home/cloudera/workspace/HiveUDFs/target/HiveUDFs-0.0.1-SNAPSHOT.jar] to
class path
```

```
Added resources: [/home/cloudera/workspace/HiveUDFs/target/HiveUDFs-0.0.1-
SNAPSHOT.jar]
```

Verify JAR is in Hive CLI Classpath

You should see your jar in the list.

```
hive> list jars;
/usr/lib/hive/lib/hive-contrib.jar
/home/cloudera/workspace/HiveUDFs/target/HiveUDFs-0.0.1-SNAPSHOT.jar
```

Create Temporary Function

It does not have to be a temporary function. You can create your own function, but just to keep things moving, go ahead and create a temporary function.

You may want to add ADD JAR and CREATE TEMPORARY FUNCTION to .hiverc file so they will execute at the beginning of each Hive session.

UDF Output

The first query strips 'ha' from string 'hadoop' as expected (2 argument evaluate() in code).  
The second query strips trailing and leading spaces as expected.

```
hive> CREATE TEMPORARY FUNCTION STRIP AS 'org.hardik.letsdobigdata.Strip';
```

```
hive> select strip('hadoop','ha') from dummy;
OK
doop
Time taken: 0.131 seconds, Fetched: 1 row(s)
hive> select strip(' hiveUDF ') from dummy;
OK
hiveUDF
```

## 22. Explain the different ways of implementing Hive metastore?

Ans: Metastore Configuration

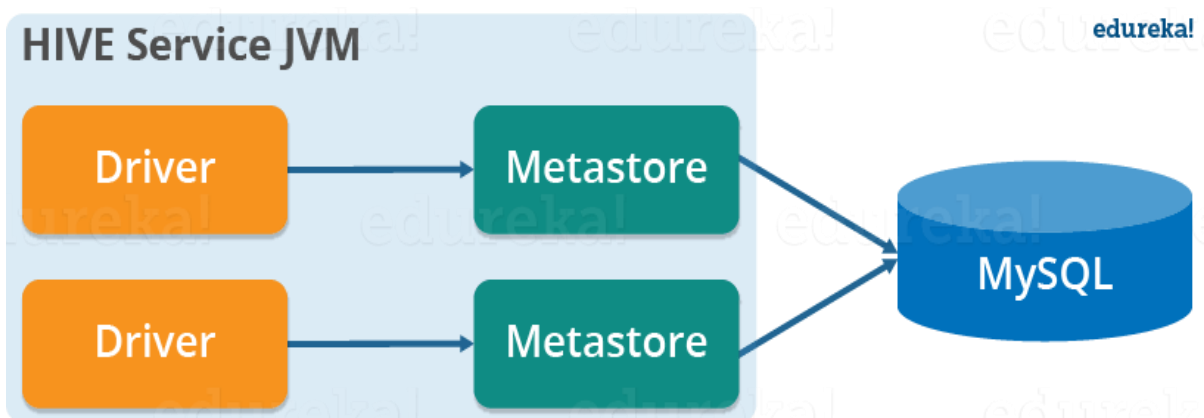
Metastore stores the meta data information using RDBMS and an open source ORM (Object Relational Model) layer called Data Nucleus which converts the object representation into relational schema and vice versa. The reason for choosing RDBMS instead of HDFS is to achieve low latency. We can implement metastore in following three configurations:

### 1. Embedded Metastore:



Both the metastore service and the Hive service runs in the same JVM by default using an embedded Derby Database instance where metadata is stored in the local disk. This is called embedded metastore configuration. In this case, only one user can connect to metastore database at a time. If you start a second instance of Hive driver, you will get an error. This is good for unit testing, but not for the practical solutions.

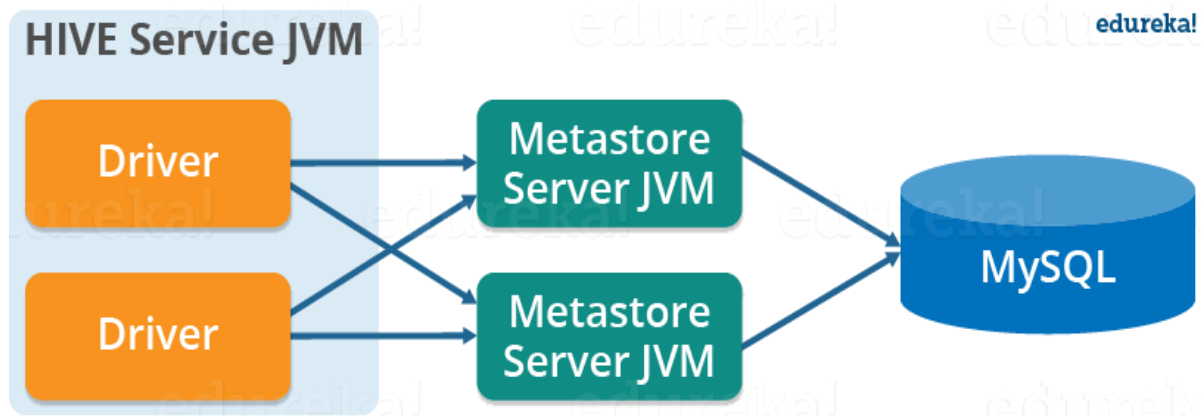
### 2. Local Metastore:



This configuration allows us to have multiple Hive sessions i.e. Multiple users can use the metastore database at the same time. This is achieved by using any JDBC compliant database like MySQL which runs in a separate JVM or a different machine than that of the Hive service and metastore service which are running in the same JVM as shown above. In general, the most popular choice is to implement a MySQL server as the metastore database.



### 3. Remote Metastore:



## 23. Differentiate between HIVE and MapReduce.

Ans: Map Reduce vs Hive

### Hadoop

- Using traditional data management systems, it is difficult to process Big Data.
- Big data by definition is the data set or a type of data that is very difficult to handle using traditional tools or conventional tools like SAS, R, Excel, SQL, etc that is used in the day-to-day life.
- Therefore, the Apache Software Foundation introduced a framework called Hadoop to solve Big Data management and processing challenges.
- Hadoop is an open-source framework to store and process Big Data in a distributed environment.
- It contains two modules, one is MapReduce and another is Hadoop Distributed File System (HDFS).

### MapReduce:

- MapReduce is a kind of distributed computing where we first divide the whole objective into various smaller tasks and then finally write a Map Reduce program where the map will carry out some computation and then the reducer will take the output of map as input and find the required final output. As we discussed in earlier sessions, HDFS is the distributed file storage. In MapReduce we saw some programs like word count program, line count program, finding the average, etc. While writing MapReduce programs, we observed that hdfs and MapReduce together increase the data processing and increases the speed of computation.
- It is a parallel programming model for processing large amounts of structured, semi-structured, and unstructured data on large clusters of commodity hardware. **HDFS:**
- Hadoop Distributed File System is a part of Hadoop framework, used to store and process the datasets. It provides a fault-tolerant file system to run on commodity hardware.

### MapReduce needs Java

- While HDFS and Map Reduce programming solves our issue with big data handling, but it is not very easy to write a MapReduce code
- One has to understand the logic of overall algorithm then convert to MapReduce format.
- MapReduce programs are written in Java

- We need to be an expert in Java to write MapReduce code for intermediate and Complex problems
- Not everyone has Java background

### Map Reduce Made Easy

- The traditional approach using Java MapReduce program for structured, semi-structured, and unstructured data.
- This way to executing map reduce is not easy
- Hive is created to make the map reduce process easy. Hive query language.
- Hive is similar to the SQL query language.
- Let's say, if we want to find the average of some data, then writing a map and reduce function in Java is difficult. In such cases writing an SQL query is much easier than writing a java code. Even if one does not know sql, learning SQL is very easy.
- So writing queries in hive will be much easier than writing a java code. Hive is like SQL on top of hadoop.
- If we write some queries in hive, it will understand our query, create the map and reduce functions, send it to hadoop distributed file system, perform the analysis on the data, and fetch the results back .

### Line count- Code

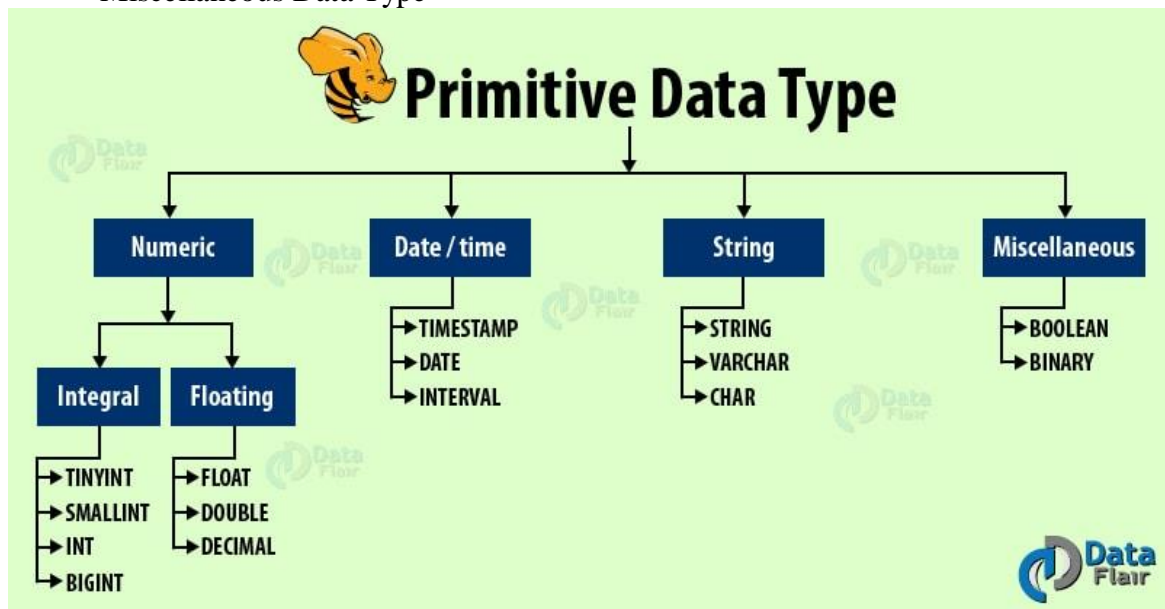
- Let's take an example of a line count code.
- In hive we just have to write "Select count(\*) from table" instead of writing a huge MapReduce code.
- For this particular line count program, we need to write many lines of code in java, whereas in hive we just have to write a single line query.

## 24. Explain various data types supported by HIVE.

Ans: Primitive Data Types in Hive

Primitive Data Types also divide into 4 types which are as follows:

- Numeric Data Type
- Date/Time Data Type
- String Data Type
- Miscellaneous Data Type



## Primitive Data Types in Hive

### *i. Numeric Data Type*

The Hive Numeric Data types also classified into two types-

- Integral Data Types
- Floating Data Types

#### **\* Integral Data Types**

Integral Hive data types are as follows-

- **TINYINT** (1-byte (8 bit) signed integer, from -128 to 127)
- **SMALLINT** (2-byte (16 bit) signed integer, from -32, 768 to 32, 767)
- **INT** (4-byte (32-bit) signed integer, from -2,147,483,648 to 2,147,483,647)
- **BIGINT** (8-byte (64-bit) signed integer, from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

#### **\* Floating Data Types**

Floating Hive data types are as follows-

- **FLOAT** (4-byte (32-bit) single-precision floating-point number)
- **DOUBLE** (8-byte (64-bit) double-precision floating-point number)
- **DECIMAL** (Arbitrary-precision signed decimal number)

### *ii. Date/Time Data Type*

The second category of Apache Hive primitive data type is Date/Time data types. The following Hive data types comes into this category-

- **TIMESTAMP** (Timestamp with nanosecond precision)
- **DATE** (date)
- **INTERVAL**

### *iii. String Data Type*

String data types are the third category under Hive data types. Below are the data types that come into this-

- **STRING** (Unbounded variable-length character string)
- **VARCHAR** (Variable-length character string)
- **CHAR** (Fixed-length character string)

### **iv. Miscellaneous Data Type**

Miscellaneous data types has two types of Hive data types-

- **BOOLEAN** (True/false value)
- **BINARY** (Byte array)

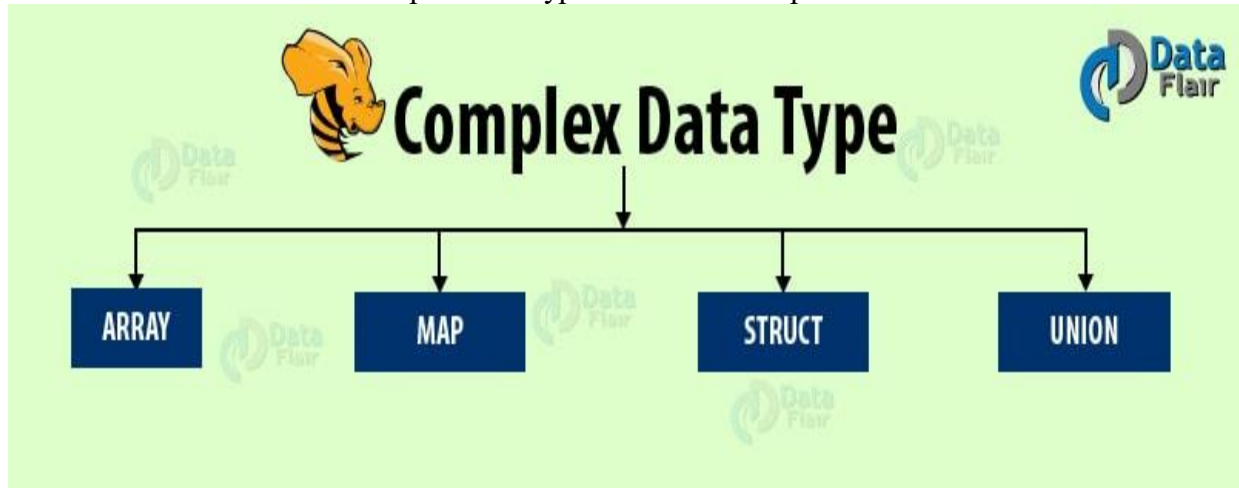
## **Complex Data Types in Hive**

In this category of Hive data types following data types are come-

- Array
- MAP
- STRUCT

- UNION

let's now discuss the Hive Complex data types with the example-



Hive Complex data types

#### *i. ARRAY*

An ordered collection of fields. The fields must all be of the same type.

**Syntax:** ARRAY<data\_type>

**E.g.** array (1, 2)

#### *ii. MAP*

An unordered collection of **key-value pairs**. Keys must be primitives; values may be any type. For a particular map, the keys must be the same type, and the values must be the same type.

**Syntax:** MAP<primitive\_type, data\_type>

**E.g.** map('a', 1, 'b', 2).

#### *iii. STRUCT*

A collection of named fields. The fields may be of different types.

**Syntax:** STRUCT<col\_name : data\_type [COMMENT col\_comment],...>

**E.g.** struct('a', 1 1.0),[b] named\_struct('col1', 'a', 'col2', 1, 'col3', 1.0)

#### *iv. UNION*

A value that may be one of a number of defined data. The value is tagged with an integer (zero-indexed) representing its data type in the union.

**Syntax:** UNIONTYPE<data\_type, data\_type, ...>

**E.g.** create\_union(1, 'a', 63)

## **25. How Hive distributes the rows into buckets?**

Ans: Buckets:

Commands:

```
CREATE TABLE table_name PARTITIONED BY (partition1 data_type, partition2
data_type,...) CLUSTERED BY (column_name1, column_name2, ...) SORTED BY
(column_name [ASC|DESC], ...) INTO num_buckets BUCKETS;
```

Now, you may divide each partition or the unpartitioned table into Buckets based on the hash function of a column in the table. Actually, each bucket is just a file in the partition directory or the table directory (unpartitioned table). Therefore, if you have chosen to divide the partitions into n buckets, you will have n files in each of your partition directory. For example, you can see the above image where we have bucketed each partition into 2 buckets. So, each partition, say CSE, will have two files where each of them will be storing the CSE student's data.

Well, Hive determines the bucket number for a row by using the formula: **hash\_function(bucketing\_column) modulo (num\_of\_buckets)**. Here, hash\_function depends on the column data type. For example, if you are bucketing the table on the basis of some column, let's say user\_id, of INT datatype, the hash\_function will be – **hash\_function(user\_id)= integer value of user\_id**. And, suppose you have created two buckets, then Hive will determine the rows going to bucket 1 in each partition by calculating: (value of user\_id) modulo (2). Therefore, in this case, rows having user\_id ending with an even integer digit will reside in a same bucket corresponding to each partition. The hash\_function for other data types is a bit complex to calculate and in fact, for a string it is not even humanly recognizable.

**Note:** If you are using Apache Hive 0.x or 1.x, you have to issue command – set hive.enforce.bucketing = true; from your Hive terminal before performing bucketing. This will allow you to have the correct number of reducer while using cluster by clause for bucketing a column. In case you have not done it, you may find the number of files that has been generated in your table directory are not equal to the number of buckets. As an alternative, you may also set the number of reducer equal to the number of buckets by using set mapred.reduce.task = num\_bucket.

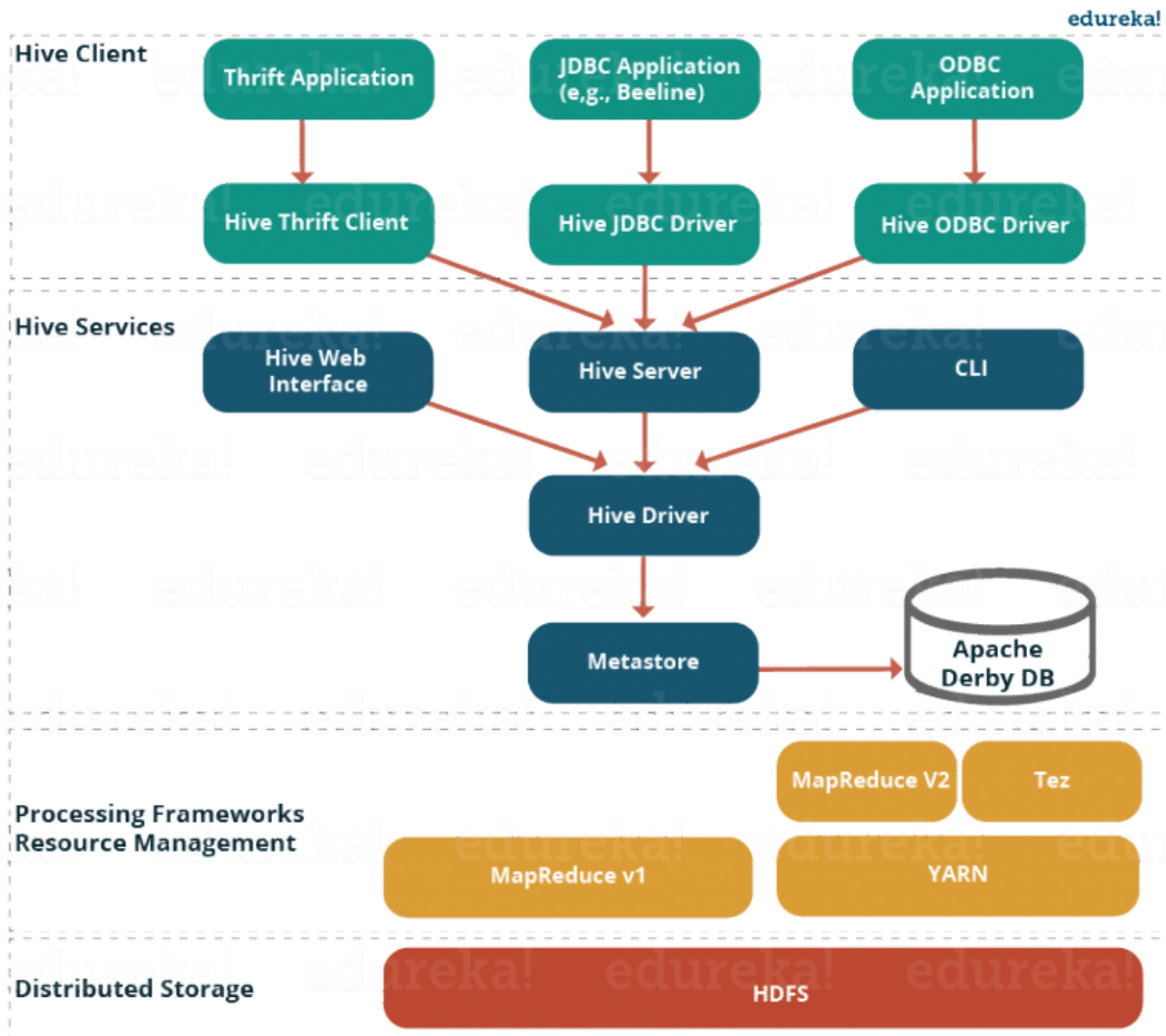
## 26. Why do we need buckets?

Ans: There are two main reasons for performing bucketing to a partition:

- A map side join requires the data belonging to a unique join key to be present in the same partition. But what about those cases where your partition key differs from join? Therefore, in these cases you can perform a map side join by bucketing the table using the join key.
- Bucketing makes the sampling process more efficient and therefore, allows us to decrease the query time.

## 27. Explain the various Hive Services.

Ans:



Hive provides many services as shown in the image above. Let us have a look at each of them:

- **Hive CLI (Command Line Interface):** This is the default shell provided by the Hive where you can execute your Hive queries and commands directly.
- **Apache Hive Web Interfaces:** Apart from the command line interface, Hive also provides a web based GUI for executing Hive queries and commands.
- **Hive Server:** Hive server is built on Apache Thrift and therefore, is also referred as Thrift Server that allows different clients to submit requests to Hive and retrieve the final result.
- **Apache Hive Driver:** It is responsible for receiving the queries submitted through the CLI, the web UI, Thrift, ODBC or JDBC interfaces by a client. Then, the driver passes the query to the compiler where parsing, type checking and semantic analysis takes place with the help of schema present in the metastore. In the next step, an optimized logical plan is generated in the form of a DAG (Directed Acyclic Graph) of map-reduce tasks and HDFS tasks. Finally, the execution engine executes these tasks in the order of their dependencies, using Hadoop.
- **Metastore:** You can think metastore as a central repository for storing all the Hive metadata information. Hive metadata includes various types of information like structure of tables and the partitions along with the column, column type, serializer and deserializer which is required for Read/Write operation on the data present in HDFS. The metastore comprises of two fundamental units:
  - A service that provides metastore access to other Hive services.
  - Disk storage for the metadata which is separate from HDFS storage.

## Syllabus part -4 Hbase:HBasics, Concepts, Clients, Example, HbaseVersus RDBMS.

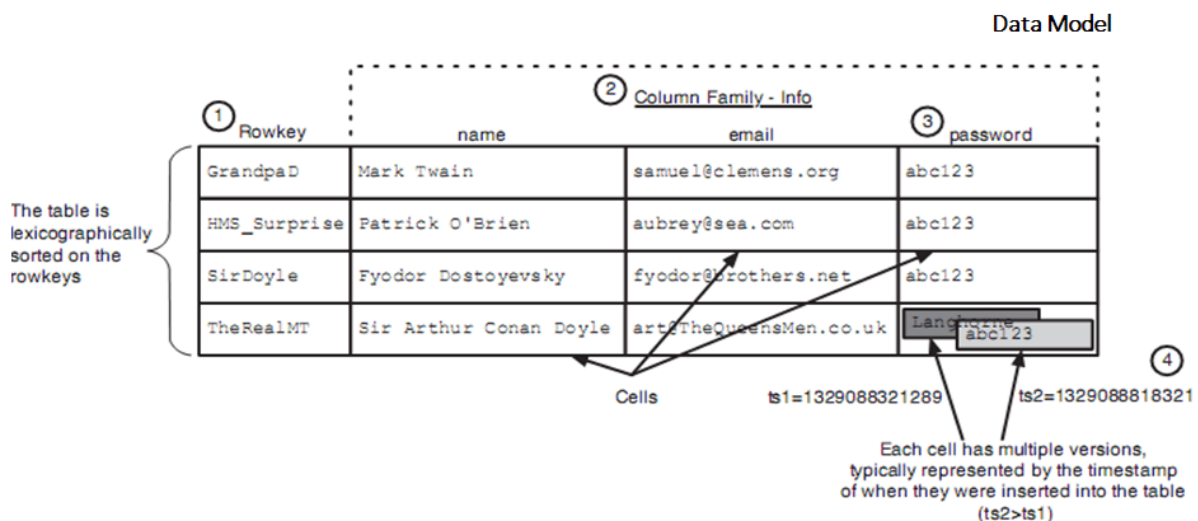
### 28 What is Hbase? Describe various features of Hbase.

Hbase is an open source and sorted map data built on Hadoop. It is column oriented and horizontally scalable.

It is based on Google's Big Table. It has set of tables which keep data in key value format. Hbase is well suited for sparse data sets which are very common in big data use cases. Hbase provides APIs enabling development in practically any programming language. It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

#### Features of Hbase

- RDBMS get exponentially slow as the data becomes large
- Expects data to be highly structured, i.e. ability to fit in a well-defined schema
- Any change in schema might require a downtime
- For sparse datasets, too much of overhead of maintaining NULL values
- Horizontally scalable: You can add any number of columns anytime.
- Automatic Failover: Automatic failover is a resource that allows a system administrator to automatically switch data handling to a standby system in the event of system compromise
- Integrations with Map/Reduce framework: All the commands and java codes internally implement Map/ Reduce to do the task and it is built over Hadoop Distributed File System.
- sparse, distributed, persistent, multidimensional sorted map, which is indexed by rowkey, column key, and timestamp.
- Often referred as a key value store or column family-oriented database, or storing versioned maps of maps.
- fundamentally, it's a platform for storing and retrieving data with random access.
- It doesn't care about datatypes (storing an integer in one row and a string in another for the same column).
- It doesn't enforce relationships within your data.
- It is designed to run on a cluster of computers, built using commodity hardware.



The coordinates used to identify data in an HBase table are 1 rowkey, 2 column family, 3 column qualifier, and 4 version.

## HBase Read

A read against HBase must be reconciled between the HFiles, MemStore & BLOCKCACHE. The BlockCache is designed to keep frequently accessed data from the HFiles in memory so as to avoid disk reads. Each column family has its own BlockCache. BlockCache contains data in form of 'block', as unit of data that HBase reads from disk in a single pass. The HFile is physically laid out as a sequence of blocks plus an index over those blocks. This means reading a block from HBase requires only looking up that block's location in the index and retrieving it from disk.

**Block:** It is the smallest indexed unit of data and is the smallest unit of data that can be read from disk. default size 64KB.

**Scenario, when smaller block size is preferred:** To perform random lookups. Having smaller blocks creates a larger index and thereby consumes more memory.

**Scenario, when larger block size is preferred:** To perform sequential scans frequently. This allows you to save on memory because larger blocks mean fewer index entries and thus a smaller index.

Reading a row from HBase requires first checking the MemStore, then the BlockCache, Finally, HFiles on disk are accessed.

## HBase Write

When a write is made, by default, it goes into two places:

- write-ahead log (WAL), HLog, and
- in-memory write buffer, MemStore.

Clients don't interact directly with the underlying HFiles during writes, rather writes goes to WAL & MemStore in parallel. Every write to HBase requires confirmation from both the WAL and the MemStore.

## HBase MemStore

- The MemStore is a write buffer where HBase accumulates data in memory before a permanent write.
- Its contents are flushed to disk to form an HFile when the MemStore fills up.
- It doesn't write to an existing HFile but instead forms a new file on every flush.
- The HFile is the underlying storage format for HBase.
- HFiles belong to a column family(one MemStore per column family). A column family can have multiple HFiles, but the reverse isn't true.
- size of the MemStore is defined in hbase-site.xml called `hbase.hregion.memstore.flush.size`.

## What happens, when the server hosting a MemStore that has not yet been flushed crashes?

Every server in HBase cluster keeps a WAL to record changes as they happen. The WAL is a file on the underlying file system. A write isn't considered successful until the new WAL entry is successfully written, this guarantees durability.

If HBase goes down, the data that was not yet flushed from the MemStore to the HFile can be recovered by replaying the WAL, taken care by Hbase framework.

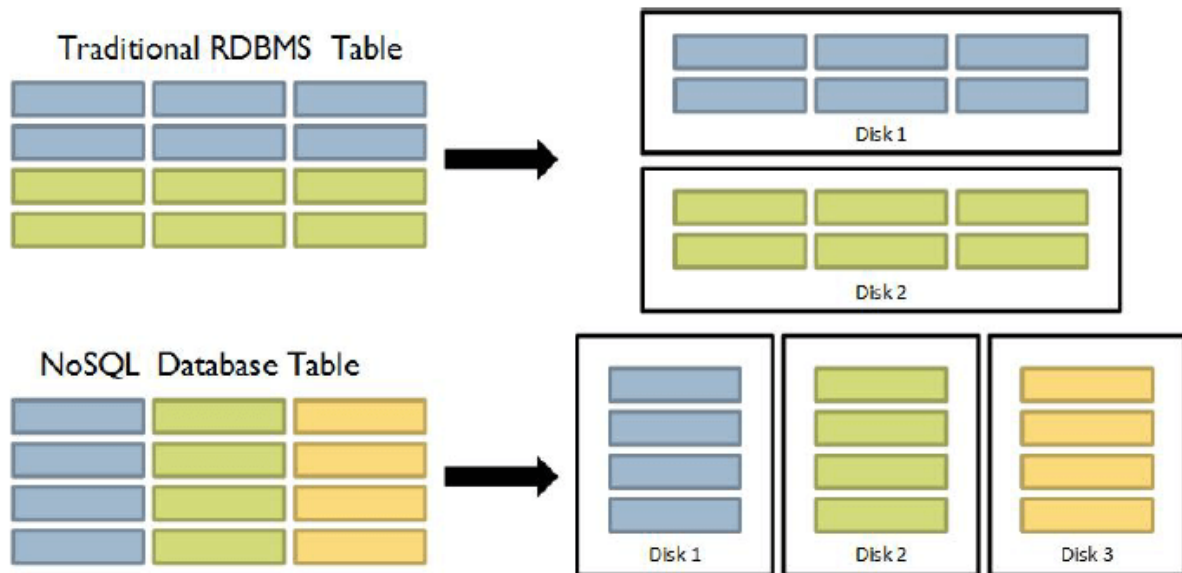


## Defferentiate between RDBMS and Hbase.

### RDBMS vs HBase

There differences between RDBMS and HBase are given below.

- Schema/Database in RDBMS can be compared to namespace in Hbase.
- A table in RDBMS can be compared to column family in Hbase.
- A record (after table joins) in RDBMS can be compared to a record in Hbase.
- A collection of tables in RDBMS can be compared to a table in Hbase..



### HBase Commands

A list of HBase commands are given below.

- **Create:** Creates a new table identified by 'table1' and Column Family identified by 'colf'.
- **Put:** Inserts a new record into the table with row identified by 'row..'
- **Scan:** returns the data stored in table
- **Get:** Returns the records matching the row identifier provided in the table
- **Help:** Get a list of commands

1. create 'table1', 'colf'
2. list 'table1'
3. put 'table1', 'row1', 'colf:a', 'value1'
4. put 'table1', 'row1', 'colf:b', 'value2'
5. put 'table1', 'row2', 'colf:a', 'value3'
6. scan 'table1'
7. get 'table1', 'row1'

### HBase Example

Let's see a HBase example to import data of a file in HBase table.

## Use Case

We have to import data present in the file into an HBase table by creating it through Java API.

Data\_file.txt contains the below data

```
1,India,Bihar,Champaran,2009,April,P1,1,5
2,India, Bihar,Patna,2009,May,P1,2,10
3,India, Bihar,Bhagalpur,2010,June,P2,3,15
4,United States,California,Fresno,2009,April,P2,2,5
5,United States,California,Long Beach,2010,July,P2,4,10
6,United States,California,San Francisco,2011,August,P1,6,20
```

The Java code is shown below

This data has to be inputted into a new HBase table to be created through JAVA API.

Following column families have to be created

1. "sample,region,time.product,sale,profit".

Column family region has three column qualifiers: country, state, city

Column family Time has two column qualifiers: year, month

## Jar Files

Make sure that the following jars are present while writing the code as they are required by the HBase.

- a. commons-logging-1.0.4
- b. commons-logging-api-1.0.4
- c. hadoop-core-0.20.2-cdh3u2
- d. hbase-0.90.4-cdh3u2
- e. log4j-1.2.15
- f. zookeeper-3.3.3-cdh3u0

## Program Code

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.client.HBaseAdmin;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.util.Bytes;

public class readFromFile {
 public static void main(String[] args) throws IOException{
 if(args.length==1)
 {
```

```

Configuration conf = HBaseConfiguration.create(new Configuration());
HBaseAdmin hba = new HBaseAdmin(conf);
if(!hba.tableExists(args[0])){
 HTableDescriptor ht = new HTableDescriptor(args[0]);
 ht.addFamily(new HColumnDescriptor("sample"));
 ht.addFamily(new HColumnDescriptor("region"));
 ht.addFamily(new HColumnDescriptor("time"));
 ht.addFamily(new HColumnDescriptor("product"));
 ht.addFamily(new HColumnDescriptor("sale"));
 ht.addFamily(new HColumnDescriptor("profit"));
 hba.createTable(ht);
 System.out.println("New Table Created");

 HTable table = new HTable(conf,args[0]);

 File f = new File("/home/training/Desktop/data");
 BufferedReader br = new BufferedReader(new FileReader(f));
 String line = br.readLine();
 int i =1;
 String rowname="row";
 while(line!=null && line.length()!=0){
 System.out.println("Ok till here");
 StringTokenizer tokens = new StringTokenizer(line, ",");
 rowname = "row"+i;
 Put p = new Put(Bytes.toBytes(rowname));
 p.add(Bytes.toBytes("sample"),Bytes.toBytes("sampleNo."),
Bytes.toBytes(Integer.parseInt(tokens.nextToken())));
 p.add(Bytes.toBytes("region"),Bytes.toBytes("country"),Bytes.toBytes(tokens.nextT
oken()));
 p.add(Bytes.toBytes("region"),Bytes.toBytes("state"),Bytes.toBytes(tokens.nextToke
n()));
 p.add(Bytes.toBytes("region"),Bytes.toBytes("city"),Bytes.toBytes(tokens.nextToke
n()));
 p.add(Bytes.toBytes("time"),Bytes.toBytes("year"),Bytes.toBytes(Integer.parseInt(to
kens.nextToken())));
 p.add(Bytes.toBytes("time"),Bytes.toBytes("month"),Bytes.toBytes(tokens.nextToke
n()));
 p.add(Bytes.toBytes("product"),Bytes.toBytes("productNo."),Bytes.toBytes(tokens.n
extToken()));
 p.add(Bytes.toBytes("sale"),Bytes.toBytes("quantity"),Bytes.toBytes(Integer.parseIn
t(tokens.nextToken())));
 p.add(Bytes.toBytes("profit"),Bytes.toBytes("earnings"),Bytes.toBytes(tokens.nextT
oken()));
 i++;
 table.put(p);
 line = br.readLine();
 }
 br.close();
 table.close();
}
else
 System.out.println("Table Already exists.Please enter another table name");
}
else
 System.out.println("Please Enter the table name through command line");
}

```

}

## **Q) Comparison between Hive, Pig, SQL.**

**Ans:**

### **# Language used**

**Hive** - Apache Hive uses HiveQL, a declarative language.

**Pig** - Pig uses Pig Latin, procedural dataflow language.

**SQL** - SQL itself is a declarative language.

### **# Operational for**

**Hive** - Hive is for Structured Data

**Pig** - Pig is for structured and semi-structured data

**SQL** - SQL deals with structured data and is for RDBMS that is a relational database management

### **# Schema**

**Hive** - Hive support schema for data insertion

**Pig** - Pig doesn't support schema

**SQL** - SQL support schema for data storage

### **# Usage**

**Hive** - We use Hive when working on structured data to query large data sets and analyse historical data

**Pig** - We use Pig as an ETL tool. Pig is faster than Hive and has many functions related to SQL.

**SQL** - We use SQL when we need frequent modification in records. SQL is used for better performance

Now, here is a brief explanation -

**Hive** - Apache Hive is a big data software we use it in writing, reading and managing huge datasets. Hive is built on Hadoop and it is an open source project to analyse query datasets. HiveQL is a language that is similar to SQL, it converts the queries into Map Reduce programmes.

**Apache Pig** - Apache Pig is a high-level data flow platform for executing Map Reduce programs of Hadoop. We use Pig for ease programming, optimization, and extensibility. If you are aware of SQL it will be easy to learn and use Pig. Many big companies such as Google, Yahoo, and Microsoft use Pig for analysing data sets out of search logs, web crawls, and click streams.

**SQL** - SQL is Structured Query Language, it is easy to learn. We use SQL to manipulate, access, modify and manage the data in the database. The data stored in the relational database systems. SQL is a fast tool for data processing and analysis.

## Big SQL: Introduction

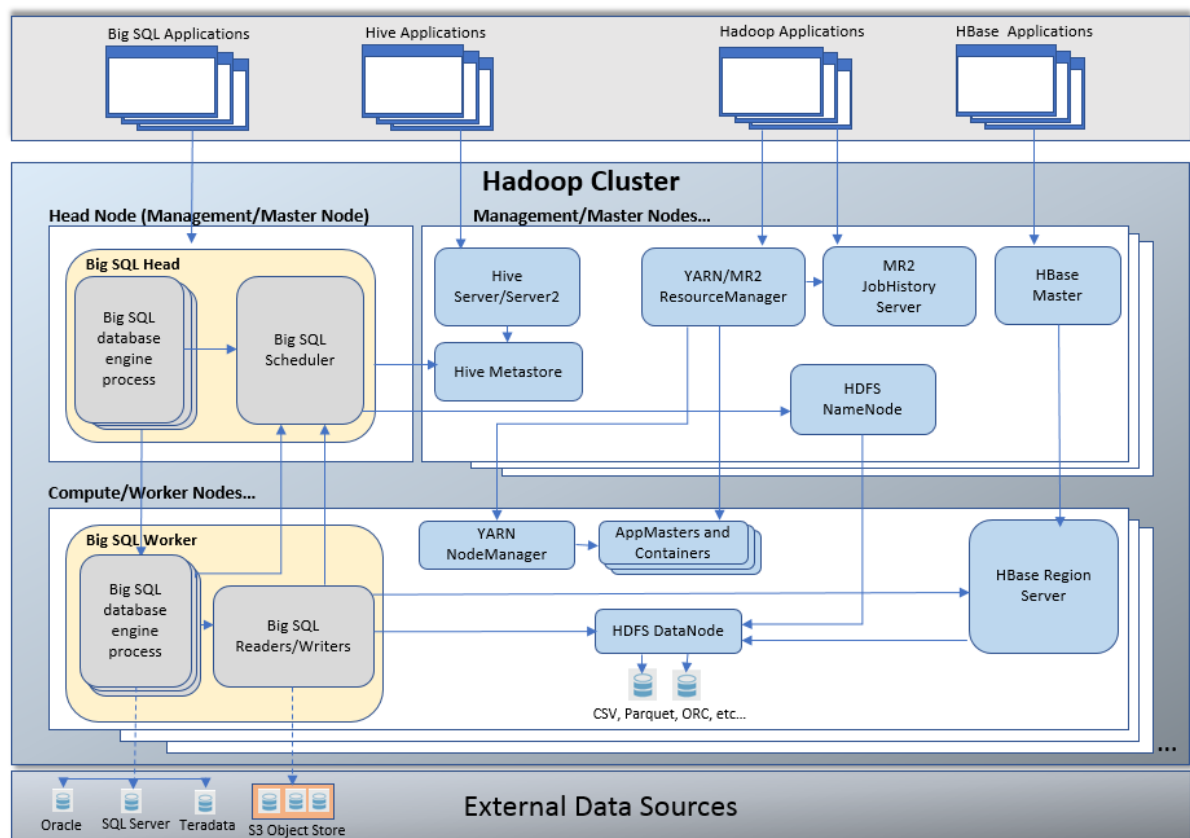
### What Big SQL ?

Big SQL is a high performance massively parallel processing (MPP) SQL engine for Hadoop that makes querying enterprise data from across the organization an easy and secure experience. A Big SQL query can quickly access a variety of data sources including HDFS, RDBMS, NoSQL databases, object stores, and WebHDFS by using a single database connection or single query for best-in-class analytic capabilities.

### Big SQL architecture:

Built on the world class IBM common SQL database technology, Big SQL is a massively parallel processing (MPP) database engine that has all the standard RDBMS features and is optimized to work with the Apache Hadoop ecosystem.

The following diagram shows how Big SQL fits within the overall Apache Hadoop architecture.



The Big SQL server or service consists of one Big SQL head (two heads in an HA configuration) that is installed on a node called the head node, and multiple Big SQL workers that are installed on nodes called worker nodes.

### Definitions

#### Big SQL server

A general term to describe the Big SQL software or the Big SQL processes. *Big SQL service* is a synonym for Big SQL server in the context of Big SQL as a service in the HDP stack.

#### Big SQL head

The set of Big SQL processes that accept SQL query requests from applications and coordinate with Big SQL workers to process data and compute the results.

### **Big SQL head node**

The physical or virtual machine (node) on which the Big SQL head runs.

### **Big SQL worker**

The set of Big SQL processes that communicate with the Big SQL head to access data and compute query results. Big SQL workers are normally collocated with the HDFS DataNodes to facilitate local disk access. Big SQL can access and process HDFS data that conforms to most common Hadoop formats, such as Avro, Parquet, ORC, Sequence, and so on. For more details about the supported data formats, see [File formats supported by Big SQL](#). The Big SQL head coordinates the processing of SQL queries with the workers, which handle most of the HDFS data access and processing.

### **Big SQL worker node**

The physical or virtual machine (node) on which the Big SQL worker runs.

### **Big SQL scheduler**

A process that runs on the Big SQL head node. The scheduler's function is to bridge the RDBMS domain and the Hadoop domain. The scheduler communicates with the Hive metastore to determine Hive table properties and schemas, and the HDFS NameNode to determine the location of file blocks. The scheduler responds to Big SQL head and worker requests for information about Hadoop data, including HDFS, HBase, object storage, and Hive metadata. For more information about the scheduler, see [Big SQL scheduler](#).

### **Big SQL metadata**

HDFS data properties such as name, location, format, and the desired relational schemas. This metadata, gathered through the scheduler, is used to enable consistent and optimal SQL processing of queries against HDFS data.

### **How Big SQL processes HDFS data**

The following steps represent a simple overview of how Big SQL processes HDFS data:

1. **Applications connect.**  
Applications connect to the Big SQL head on the head node.
2. **Queries are submitted.**  
Queries submitted to the Big SQL head are compiled into optimized parallel execution plans by using the IBM common SQL engine's query optimizer.
3. **Plans are distributed.**  
The parallel execution plans are then distributed to Big SQL workers on the worker nodes.
4. **Data is read and written.**  
Workers have separate local processes called native HDFS readers and writers that read or write HDFS data in its stored format. A *Big SQL reader* is comprised of Big SQL processes that run on the Big SQL worker nodes and read data at the request of Big SQL workers. Similarly, a *Big SQL writer* is comprised of Big SQL processes that run on the Big SQL worker nodes and write data at the request of Big SQL workers. The workers communicate with these native readers or writers to access HDFS data when they process execution plans that they receive from the Big SQL head. For more information, see [Big SQL readers and writers](#).
5. **Predicates are applied.**  
Native HDFS readers can apply predicates and project desired columns to minimize the amount of data that is returned to workers.

### **Big SQL features**

Big SQL features include easy-to-use tools, flexible security options, strong federation and performance capabilities, and a massively parallel processing (MPP) SQL engine that provides powerful SQL processing features.

- **Administration tools**

With Big SQL, you can manage your system with the Ambari dashboard, and you can manage your databases with Data Server Manager (DSM).

- **Visualization tools**

Big SQL integrates with easy-to-use tools for data visualization: Zeppelin notebooks, Data Science Experience, Tableau, and Cognos.

- **Security features**

Big SQL includes a powerful SQL processing engine, works in a platform with a data warehouse based on Apache Hive, accesses the DSM console using Apache Knox, and can perform authorization and auditing using the Apache Ranger framework.

- **Federation capabilities**

With Big SQL you can efficiently query data on Hadoop and also combine data spread around different enterprise data warehouses. The **federation capability** of Big SQL lets you query against and combine with Hadoop data, as well as letting you push down predicates. Not all data moves back and forth between the systems; only the results of the predicates are sent back to combine with Hadoop data.

- **Performance capabilities**

Big SQL provides superior SQL-on-Hadoop performance to optimize data ingestion and query performance for your enterprise. Big SQL is not only fast and efficient, but more importantly can successfully execute the most demanding queries on big data. Big SQL provides a robust runtime environment and is compliant with SQL standards.

- **SQL processing features**

The Big SQL massively parallel processing (MPP) SQL engine provides you with several SQL processing features.